(54) Title: APPARATUS AND METHOD FOR REAL-TIME VOLUME PROCESSING AND UNIVERSAL 3D RENDERING

(57) Abstract

An apparatus and method for real–time volume processing and universal three–dimensional rendering (10). The apparatus includes a plurality of three–dimensional memory units; at least one pixel bus for providing global horizontal communication; a plurality of rendering pipelines; at least one geometry bus; and a control unit. The plurality of rendering pipelines each preferably include hardware for interpolation, shading, FIFO buffering, communication and lookup tables. The apparatus of the present invention may be coupled to a geometry pipeline (18) for mixing surfaces, images and volumes together in a single image. A method for performing volumetric ray casting of a 3D volume includes the steps of calculating a distance along a major projection axis from a predefined viewpoint; dividing the volume into a plurality of consecutive regions having exponentially increasing bounds; casting a plurality of rays from the viewpoint through the volume; either merging two or more rays or splitting one or more rays at the region boundaries; and repeating the ray casting and merging/splitting steps until the entire volume has been processed. The apparatus and methods of the present invention achieve true real–time performance for high–resolution volume rendering, mixing surfaces and volumes in a single image operations, including texture mapping and image–based rendering.

APPARATUS AND METHOD FOR REAL-TIME VOLUME PROCESSING AND UNIVERSAL 3D RENDERING

STATEMENT OF GOVERNMENT RIGHTS

This invention was made with Government support under grant MIP9527694 awarded by the National Science Foundation and under grant N000149710402 awarded by the Office of Naval Research. The Government has certain rights in the invention.

BACKGROUND OF THE INVENTION

This application claims priority to U.S. Provisional Patent Application Serial No. 60/092,977, filed on July 16, 1998.

5

Field of the Invention

The present invention relates generally to three-dimensional (3D) graphics and volume visualization, and more particularly relates to an apparatus and method for real time volume processing and universal three-dimensional rendering.

10

Description of the Prior Art

Computer rendering is the process of transforming complex information into a format which is comprehensible to human thought, while maintaining the integrity and accuracy of the information. Volumetric data, which consists of information relating to three-dimensional phenomena, is one species of complex information that can benefit from improved image rendering techniques. The process of presenting volumetric data, from a given viewpoint, is commonly referred to as volume rendering.

15

2

Volume visualization is a vital technology in the interpretation of the great
amounts of volumetric data generated by acquisition devices (e.g., biomedical
scanners), by supercomputer simulations, or by synthesizing geometric models using
volume graphics techniques. Of particular importance for manipulation and display of
5   volumetric objects are the interactive change of projection and rendering parameters,
real-time display rates, and in many cases, the possibility to view changes of a
dynamic dataset over time, called four-dimensional (4D) visualization (i.e.,
spatial-temporal), as in the emerging integrated acquisition visualization systems.

10   A volumetric dataset is commonly represented as a 3D grid of volume
elements (voxels), often stored as a full 3D raster (i.e., volume buffer) of voxels.
Volume rendering is one of the most common techniques for visualizing the 3D scalar
field of a continuous object or phenomenon represented by voxels at the grid points of
the volume dataset, and can be accomplished using two primary methods: object-order
15   methods and image-order methods. Using an object-order approach, the contribution
of each voxel to the screen pixels is calculated, and the combined contribution yields
the final image. Using an image-order approach, sight rays are cast from screen pixels
through the volume dataset, and contributions of voxels along these sight rays are
used to evaluate the corresponding pixel values.

20

Over the past three decades graphics systems have evolved duofold: from
primarily two-dimensional (2D) to 3D and 4D (space and time), and from vector
graphics to raster graphics, where the vector has been replaced by the polygon as the
basic graphics primitive. This has led to the proliferation of polygon-based geometry
25   engines, optimized to display millions of triangles per second. In such systems,
however, triangle facets only approximate the shape of objects. Still, the 3D
polygon-based graphics market continues to boom, and has become one of the hottest
arenas of the personal computer (PC) industry.

30   In response to emerging demands placed on traditional graphics systems,
various techniques have been devised to handle and display discrete imagery in order

3

to enhance visual realism of the geometric model, as well as enhance or replace object

shape and structure. Among these techniques include 2D texture and photo mapping,

environment mapping, range images for image-based rendering, 2D mip-mapping,

video streams, 3D volumes, 3D mip-mapping, 4D light fields and lumigraphs, and

5   five-dimensional (5D) plenoptic functions. All these techniques require some sort of

dimensionality-based interpolation (bilinear, trilinear, quadrilinear, etc.) between

discrete pixels, texels, voxels, or n-oxels.

10   Special purpose computer architectures and methods for volume visualization

are known in the art. Traditional methods of volume visualization typically operate

by scanning through a volume dataset in a sequential manner in order to provide an

accurate representation of an object. For example, Cube-4, an architecture developed

by Dr. Arie Kaufman, Ingmar Bitter and Dr. Hanspeter Pfister, some of whom are also

named inventors in the present application, is a special purpose scalable volume

15   rendering architecture based on slice-parallel ray-casting. Cube-4 is capable of

delivering true real-time ray-casting of high resolution datasets (e.g., 1024^3 16-bit

voxels at 30 Hertz frame rate). However, Cube-4 cannot deliver such real-time

performance for perspective projections. Presently, in known prior art rendering

systems, the use of perspective projections either increases the rendering time or

20   decreases the projected image quality. Additionally, prior architectures do not

provide the ability to combine volumes and geometries into a single image.

Referring now to Figure 1, a conventional volume visualization system 1 is

shown. As illustrated in Figure 1, the volume data is stored on a disk 2 and loaded

25   into memory 4 before rendering. A Central Processing Unit (CPU) 6 then computes

the volume rendered image from the data residing in memory 4. The final image is

written to a frame buffer 8, which is typically embedded on a graphics card, for

displaying on a monitor 9 or similar display device.

30   The present invention, therefore, is intended to provide a method and

apparatus which significantly enhances the capabilities of known methods and

4

apparatus to the extent that it can be considered a new generation of imaging data processing.

Other and further objects will be made known to the artisan as a result of the present disclosure, and it is intended to include all such objects which are realized as a result of the disclosed invention.

SUMMARY OF THE INVENTION

The present invention is tantamount to a departure from the prior art because of the all-encompassing new characteristics. An apparatus, in accordance with the present invention, for real-time volume processing and universal three-dimensional (3D) rendering includes one or more three-dimensional (3D) memory units; at least a first pixel bus; one or more rendering pipelines; one or more geometry busses; and a control unit. The apparatus is responsive to viewing and processing parameters which define a viewpoint, and the apparatus generates a 3D volume projection image from the viewpoint. The projected image includes a plurality of pixels.

The 3D memory units store a plurality of discrete voxels, each of the voxels having a location and voxel data associated therewith. The voxels together form a volume dataset, and the viewing and processing parameters define at least one face of the volume dataset as the base plane of the volume dataset as well as first and last processing slices of the volume dataset. The control unit initially designates the first processing slice as a current slice of sample points, and controls sweeping through subsequent slices of the volume dataset as current slices until the last processing slice is reached.

Each of the plurality of rendering pipelines is vertically coupled to both a corresponding one of the plurality of 3D memory units and the at least first pixel bus, and each of the rendering pipelines has global horizontal communication preferably

5

with at most its two nearest neighbors. The rendering pipelines receive voxel data from the corresponding 3D memory units and generate a two-dimensional (2D) base plane image aligned with a face of the volume dataset. The geometry I/O bus provides global horizontal communication between the plurality of rendering

5   pipelines and a geometry engine, and the geometry I/O bus enables the rendering of geometric and volumetric objects together in a single image.

The apparatus and methods of the present invention surpass existing 3D volume visualization architectures and methods, not only in terms of enhanced

10   performance, image rendering quality, flexibility and simplicity, but in terms of the ability to combine both volumes and surfaces (particularly translucent) in a single image. The present invention provides flexible, high quality, true real-time volume rendering from arbitrary viewing directions, control of rendering and projection parameters, and mechanisms for visualizing internal and surface structures of high-

15   resolution datasets. It further supports a variety of volume rendering enhancements, including accurate perspective projection, multi-resolution volumes, multiple overlapping volumes, clipping, improved gradient calculation, depth cuing, haze, super-sampling, anisotropic datasets and rendering of large volumes.

20   The present invention is more than a mere volume rendering machine; it is a high-performance interpolation engine, and as such, it provides hardware support for high-resolution volume rendering and acceleration of discrete imagery operations that are highly dependent on interpolation, including 2D and 3D texture mapping (with mip-mapping) and image-based rendering. Furthermore, the apparatus and methods

25   of the present invention, coupled with a geometry engine, combine volumetric and geometric approaches, allowing users to efficiently model and render complex scenes containing traditional geometric primitives (e.g., polygonal facets), images and volumes together in a single image (defined as universal 3D rendering).

30   The apparatus of the present invention additionally provides enhanced system flexibility by including various global and local feedback connections, which adds the

6

ability to reconfigure the pipeline stages to perform advanced imagery operations, such as imaging warping and multi-resolution volume processing. Furthermore, the present invention accomplishes these objectives in a cost-effective manner.

It is an advantage of the present invention to provide a method and apparatus which, when coupled to a geometry engine, enables the mixing of geometries and volumes together in a single image.

It is another advantage of the present invention to provide a method and apparatus for efficiently rendering multiple overlapping volumetric objects.

It is also an advantage of the present invention to provide an apparatus having the ability to reconfigure selective pipeline stages to perform advanced volume rendering functionalities.

It is a further advantage of the present invention to provide a method and apparatus which supports enhanced imaging operations that are highly dependent on interpolation.

It is still a further advantage of the present invention to provide a method and apparatus which accelerate the processing of large volume datasets.

It is another advantage of the present invention to provide either higher-quality or faster perspective projections than existing special purpose hardware volume visualization systems.

It is still another advantage of the present invention to provide a method and apparatus which are readily capable of improving the quality of images with enhancements to gradient estimation and interpolation units.

7

It is yet another advantage of the present invention to provide an apparatus which is capable of implementing image warping in hardware.

It is yet a further advantage of the present invention to provide a method and apparatus which overcome the inherent disadvantages of known volume visualization systems.

These and other features and advantages of the present invention will become apparent from the following detailed description of illustrative embodiments thereof, which is to be read in connection with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram of a conventional volume visualization system.

Figure 2 is a conceptual block diagram illustrating a universal three-dimensional rendering system formed in accordance with one embodiment of the present invention.

Figure 3 is a simplified block diagram of the Cube-5 unit of Figure 2 illustrating a preferred implementation of the present invention.

Figure 4 is a functional block diagram depicting an overview of the universal three-dimensional rendering architecture formed in accordance with one embodiment of the present invention.

Figure 5 is a functional block diagram illustrating a unit rendering pipeline formed in accordance with one embodiment of the present invention.

8

Figure 6A is a graphical representation showing how 32 bits of texel data is stored for 2×2 neighborhood in a miniblock of 16-bit voxels, in accordance with a preferred method of the present invention.

Figure 6B depicts a tabular comparison of voxel storage and texel storage for the example of Figure 6A.

Figure 7 illustrates special parallel preserving scanlines in the source and target images in accordance with a preferred forward image warping method of the present invention.

Figure 8 is a graphical representation illustrating a method for determining the parallel preserving scanline direction in accordance with the preferred forward image warping method of the present invention.

Figure 9 is two-dimensional graphical representation of an example illustrating pixel read templates in the source image for performing scanline processing, in accordance with the preferred forward image warping method of the present invention.

Figure 10 is two-dimensional graphical representation of the example of Figure 9 illustrating a bilinear interpolation of samples for performing scanline processing, in accordance with the preferred forward image warping method of the present invention.

Figure 11 is two-dimensional graphical representation of a linear interpolation on samples to obtain pixel values for performing target pixel correction, in accordance with a preferred method of the present invention.

9

Figure 12 is a graphical representation illustrating the calculation of an anisotropic filter footprint for performing antialiasing in accordance with a preferred forward image warping method of the present invention.

Figure 13 is a graphical representation illustrating the splatting of source pixels onto the target samples, in accordance with the preferred forward image warping method of the present invention.

Figure 14 depicts an example of a y-slice shear for performing three-dimensional rotation by two-dimensional slice shear decomposition, in accordance with one method of the present invention.

Figure 15 depicts an example of an x-beam shear for performing three-dimensional rotation by two-dimensional beam shear decomposition, in accordance with another method of the present invention.

Figure 16 depicts an example of an x-slice-y-beam shear for performing three-dimensional rotation by two-dimensional slice-beam shear decomposition, in accordance with still another method of the present invention.

Figure 17 depicts an example of a three-dimensional x-beam shear for performing three-dimensional rotation by three-dimensional beam shear decomposition, in accordance with yet another method of the present invention.

Figure 18A illustrates a conventional undersampling method for performing perspective projections.

Figure 18B illustrates a conventional oversampling method for performing perspective projections.

10

Figure 19 illustrates an adaptive perspective ray-casting method for performing perspective volumetric projections, in accordance with a preferred form of the present invention, wherein a view frustum is divided into regions based on exponentially increasing distance from a viewpoint.

5

Figure 20A is a graphical representation illustrating the splitting/merging of rays at exponential region boundaries, in accordance with the preferred adaptive perspective ray-casting method of the present invention.

10

Figure 20B is a graphical representation illustrating the effective filter weights for ray segments A, B and C of the adaptive perspective ray-casting method example of Figure 20A.

Figure 21 illustrates an example of the weights for a two-dimensional filter of size ±2 samples, in accordance with a preferred form of the present invention.

15

Figure 22 is a graphical representation illustrating an example of a of the adaptive perspective ray-casting method of the present invention, wherein a 7^3 volume is three voxel units from the viewpoint.

20

Figure 23 is a pseudo-code representation of a preferred method for performing Exponential-Regions Perspective back-to-front projection of a volume, in accordance with one form of the present invention.

25

Figure 24 illustrates an example of the Exponential-Regions Perspective ray casting method of the present invention across two regions.

Figure 25 depicts an example of the preferred weights for performing a 3^3 symmetric approximation of the x-component of a Sobel gradient filter, in accordance with one embodiment of the present invention.

30

Figure 26 is a graphical representation illustrating a method for mixing geometric objects and volumes in a single image in accordance with one form of the present invention.

5   Figure 27 is a graphical representation of a method for clipping triangles to thin slab boundaries in accordance with one form of the present invention.

Figure 28 is a graphical representation of a method for bucket sorting translucent polygons in accordance with a preferred form of the present invention.

10   Figure 29 is a graphical representation of a method, in accordance with one form of the present invention, for creating sheared viewing geometry by pre-warping the polygon footprints.

15   Figure 30 is a graphical representation of a Cube-5 pipeline, formed in accordance with one form of the present invention, illustrating an SRAM composite buffer included therein.

Figure 31 is a graphical representation of a conventional graphics accelerator, 20   conceptually illustrating the interfaces between the texture memory, frame buffer and geometry pipeline.

Figure 32 is a graphical representation illustrating one embodiment of the present invention employing a dual-use DRAM frame buffer connecting a geometry 25   pipeline with the Cube-5 volume rendering pipeline of the present invention.

Figure 33 is a block diagram illustrating memory interfaces for each Cube-5 pipeline including a coxel FIFO queue, in accordance with one form of the present invention.

30

Figure 34 is a graphical representation of a RGBα coxel layout onto eight DRAM chips, formed in accordance with a preferred embodiment of the present invention.

Figure 35 is a partial block diagram representation of an embedded DRAM chip implementation of run-length encoding (RLE) frame buffer hardware, formed in accordance with one form of the present invention.

Figure 36 is a pseudo-code representation showing processing occurring in the RLE hardware of Figure 35, in accordance with one form of the present invention.

Figure 37 is a graphical representation of a preferred embodiment of the present invention illustrating a RLE frame buffer connecting a geometry pipeline to an SRAM compositing buffer included in the Cube-5 pipeline.

Figure 38 illustrates a density profile of an oriented box filter taken along a line from the center of a solid primitive outward, perpendicular to the surface, in accordance with one form of the present invention.

Figure 39 illustrates a density profile of an oriented box filter taken along a line perpendicular to a triangle surface primitive, in accordance with another form of the present invention.

Figure 40 depicts a two-dimensional illustration of seven voxelization regions for a triangle primitive, in accordance with a preferred embodiment of the present invention.

Figure 41 is a pseudo-code representation of a method for computing the distance from a plane, in accordance with one form of the present invention.

13
Figure 42 is a block diagram representation illustrating an overview of a
hardware voxelization pipeline, formed in accordance with one embodiment of the
present invention.

Figure 43 is a block diagram depicting a distance unit which incrementally
computes the distance of a current voxel from a desired plane, in accordance with one
form of the present invention.

Figure 44 is a top view graphical representation illustrating a preferred method
for performing image-based rendering in accordance with one form of the present
invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The apparatus and methods of the present invention are capable of processing
data and supporting real-time visualization of high resolution voxel-based data sets.
The present invention is a universal three-dimensional (3D) rendering system
delivering enhanced volume rendering in addition to the integration of imagery (e.g.,
volumes, textures and images) with geometry (e.g., polygons). The apparatus and
methods are designed for use as a voxel-based system as described in the issued
patents and pending applications of Dr. Arie Kaufman, a named inventor in this
application, including "Method of Converting Continuous Three-Dimensional
Geometrical Representations Into Discrete Three-Dimensional Voxel-Based
Representations Within a Three-Dimensional Voxel-Based System", which issued on
August 6, 1991, as U.S. Patent No. 5,038,302; "Method of Converting Continuous
Three-Dimensional Geometrical Representations of Polygonal Objects Into Discrete
Three-Dimensional Voxel-Based Representations Thereof Within a Three-
Dimensional Voxel-Based System", which issued on January 22, 1991, as U.S. Patent
No. 4,987,554; "Method and Apparatus for Storing, Accessing, and Processing Voxel-
Based Data", which issued on January 15, 1991, as U.S. Patent No. 4,985,856;
"Method of Converting Continuous Three-Dimensional Geometrical Representations
of Quadratic Objects Into Discrete Three-Dimensional Voxel-Based Representations

14

Thereof Within a Three-Dimensional Voxel-Based System", which was filed on May

4, 1989, as U.S. Serial No. 07/347,593, which was abandoned in favor of U.S. Serial

No. 08/031,599, filed on March 15, 1993 as a continuation application of the '593

application; "Method and Apparatus for Generating Arbitrary Projections of Three-

Dimensional Voxel-Based Data", which issued on March 31, 1992 as U.S. Patent No.

5   5,101,475; "Method and Apparatus for Real-Time Volume Rendering From An

Arbitrary Viewing Direction", which was filed on July 26, 1993 as U.S. Serial No.

08/097,637; "Method and Apparatus For Generating Realistic Images Using a

Discrete Representation", which was filed on March 20, 1992 as U.S. Serial No.

07/855,223; and "Apparatus and Method for Parallel and Perspective Real-Time

10   Volume Visualization", which was filed on August 1, 1997 as U.S. Serial No.

08/910,575. The entire disclosure of each of these references is incorporated herein

by reference.

15   Figure 2 illustrates a conceptual view of a universal 3D rendering system 10

formed in accordance with one embodiment of the present invention. Applications 12

which display collections of renderable objects are preferably split by an Applications

Program Interface (API) 14 into appropriate imagery and geometry representations.

These representations are subsequently processed by an imagery unit 16 and a

20   geometry unit 18, respectively, which are illustrated generally as functional blocks.

The imagery unit 16 preferably includes a plurality of imagery pipelines and the

geometry unit 18 preferably includes a plurality of geometry pipelines (not shown) for

rendering the imagery and geometry representations, respectively. The rendered

outputs of the imagery unit 16 and the geometry unit 18 are subsequently combined in

25   a blending unit 20 to generate a single baseplane image. This baseplane image may

preferably be transformed by a warp unit 22 to a final projection plane for display.

        Figure 3 illustrates one implementation of the Cube-5 volume visualization

system of the present invention. As shown in Figure 3, the system preferably includes

30   one or more three-dimensional memory units 24, with each 3D memory unit 24

vertically coupled to an input bus 26 and a corresponding Cube-5 chip 28. A plurality

of Cube-5 chips 28 are shown connected to a frame buffer pixel bus 34. Furthermore,

15

the system 10 of the present invention preferably interfaces to at least one
conventional geometry engine 30 and a host computer 32, both operatively coupled
between the input bus 26 and the frame buffer pixel bus 34 for communicating with
the Cube-5 apparatus of the present invention.

Referring now to Figure 4, the apparatus of the present invention 10 includes a
plurality of 3D memory units 24 which are preferably connected to an imagery input
bus 26, providing global horizontal communication between the 3D memory units 24.
The volume dataset is commonly represented as a regular grid of volume elements, or
voxels, often stored as a full 3D raster (i.e., volume buffer). This volume dataset is
preferably distributed across the 3D memory units 24. With a skewed distribution, the
present invention allows conflict-free access to complete beams (i.e., rows) of voxels
parallel to any of the major axes, thereby reducing the memory-processor bandwidth
bottleneck. As illustrated in Figure 4, for streaming video or four-dimensional (4D)
volume data through the system 10, each 3D memory unit 24 is preferably connected
to a dedicated real-time input 36. By providing a dedicated connection to a real-time
input source, the memory-processor bandwidth bottleneck is further reduced.

The universal 3D rendering system 10 of the present invention further includes
a plurality of rendering pipelines, shown as functional blocks of Cube-5 units 38 in
Figure 4. Each rendering pipeline 38 is connected to a corresponding 3D memory
unit 24 and preferably has horizontal communication with at least preferably its two
nearest neighbors. The Cube-5 units 38 read from their dedicated 3D memory units
24 and produce a two-dimensional (2D) baseplane image. This baseplane image,
which contains a plurality of composited pixels generated by the Cube-5 units 38, is
preferably distributed across a plurality of two-dimensional (2D) memory units 40.
Each of the plurality of 2D memory units 40 is preferably connected to both a
corresponding Cube-5 pipeline unit 38 and a baseplane pixel bus 42 which provides
global horizontal communication between 2D memory units 40.

Preferably, the present invention includes a plurality of warp units 44
connected to the baseplane pixel bus 42. The warp units 44 assemble and transform

16

(i.e., warp) the baseplane image stored in the plurality of 2D memory units 40 onto a user-defined image plane. Although the present invention contemplates using a single warp unit 44 (e.g., in order to reduce the costs or overhead of the hardware), the use of a plurality of warp units 44 is desirable to accelerate image transformations.

5   The output of each of the warp units 44 is preferably connected to a frame buffer pixel bus 34 which provides global horizontal communication between warp units 44. Reading the source pixels over the baseplane pixel bus 42 and writing the final image pixels over the frame buffer pixel bus 34 preferably happens concurrently

10   in order to allow greater system throughput. Although not a preferred architecture, the present invention also contemplates sequential reading and writing by the warp units 44. In this manner, only one pixel bus may be required, assuming the one pixel bus offers sufficient bandwidth for real-time image transfer for a full screen image.

15   With continued reference to Figure 4, the present invention preferably includes a geometry input bus 46 and a geometry output bus 48, although it is contemplated to combine the two busses into a single geometry input/output bus of sufficient bandwidth for real-time imaging. The geometry input and output busses 46 and 48 are preferably connected to the inputs and outputs of the Cube-5 units 38 respectively

20   and provide for the unique coupling of at least one geometry pipeline or engine (not shown) to the present system 10. The architecture of the present invention, coupled with a geometry engine via the geometry busses 46 and 48, supports the integration of imagery, such as volumes and textures, with geometries, such as polygons and surfaces. This mixing of geometric data with volumetric objects is a powerful feature

25   which is unique to the present invention.

           Referring now to Figure 5, there is illustrated a block diagram depicting the functional stages of one of the plurality of Cube-5 rendering pipelines (reference number 38 in Figure 4), formed in accordance with one embodiment of the present

30   invention. As shown in Figure 5, each rendering pipeline 52 preferably includes four types of processing units: a trilinear interpolation unit (*TriLin*) 54, a gradient estimation unit (*Gradient*) 56, a shading unit (*Shader*) 58 and a compositing unit

17

(*Compos*) 60. Each of these rendering pipeline stages is described in detail in the

prior issued patents and pending applications of Arie Kaufman relating to prior Cube

volume visualization architectures (listed above) and are therefore only briefly

discussed herein below.

5

As discussed above in reference to the 3D memory units 24, the volume

dataset is stored as a regular grid of voxels distributed across the 3D memory units 24

in a skewed fashion, with each Cube-5 unit 38 connected to a corresponding 3D

memory unit 24 (see Figure 4). Voxels of the same skewed beam are preferably

10   fetched and processed in parallel, distributed across all Cube-5 units 38. Consecutive

slices of the volume dataset parallel to a predefined baseplane (i.e., parallel to a face

of the volume dataset which is most perpendicular to a predefined view direction) are

preferably traversed in scanline order. Referring again to Figure 5, an address

generation and control unit 62 preferably generates the addresses for access into the

15   3D memory unit 24. The address generation and control unit 62 additionally

designates a first processing slice as the current processing slice and controls

sweeping through subsequent slices of the volume dataset until the final slice has been

processed.

20   The trilinear interpolation unit 54 computes a new slice of interpolated sample

values between two processing slices. It is contemplated by the present invention that

the trilinear interpolation function may alternatively be performed as a sequence of

linear or bilinear interpolations.

25   The gradient estimation unit 56 preferably computes central difference

gradients using volume data from multiple slices of the volume dataset. Utilizing the

central difference gradients generated by the gradient estimation unit 56, sample

points of the current processing slice are subsequently shaded by the shading unit 58.

The shading unit 58 preferably uses the samples and gradients as indices into one or

30   more look-up tables (LUTs), preferably residing in each shading unit 58, which store

material color and intensity information. The material color table is dataset-type

dependent, while the color intensity table is based on a local illumination model, as

18

known by those skilled in the art. In simple terms, the multiplication of color and intensity yields a pixel color for each sample which is used in the compositing unit 60 to composite such color with the previously accumulated pixels along each sight ray.

5   With reference again to Figure 4, data for computing the next sample along a continuous sight ray may reside on a neighboring Cube-5 unit 38. In this case, the nearest-neighbor connections between Cube-5 units 38 are preferably used to transfer the necessary data to the appropriate Cube-5 unit 38, which will continue to process that particular sight ray. When compositing has been completed, the composited

10   pixels (i.e., baseplane pixels) are preferably stored in the corresponding 2D memory unit 40 connected to the Cube-5 unit pipeline 38. The baseplane pixels, which form the baseplane image, are subsequently read from the 2D memory units 40, via the baseplane pixel bus 42, and assembled by the warp units 44. The warp units 44 additionally transform the baseplane image to the final projection plane image.

15

          Referring to Figure 5, the delay of data required for the trilinear interpolation unit 54 and gradient estimation unit 56 is preferably achieved by inserting one or more first-in-first-out (FIFO) units 64 into the pipeline data path prior to being processed by the trilinear interpolation 54 and the gradient estimation 56 units. The FIFO unit(s) 64

20   may be implemented as, for example, random access memory (RAM), preferably embedded on the Cube-5 chip. The introduction of a predetermined delay may be particularly important when simultaneously processing beams of multiple slices of the volume dataset, thereby requiring more computation time between slices.

25   A compositing buffer (*Compos Buffer*) 74 operatively coupled to a bilinear interpolation unit (*BiLin*) 72 essentially provides a one slice FIFO. The bilinear interpolation unit 72 preferably interpolates to obtain values between voxels as needed for texture mapping. For volume rendering, *BiLin* 72 preferably uses only weights of 0.0 or 1.0 which selects one of the corner voxels of the volume dataset (determined by

30   *Select x* and *Select y*). It just moves the ray data, if the ray crosses pipelines. Just a mux [?] for x and y would be enough for volume rendering, but bilinear interpolation is preferred because of texture mapping.

19

The Cube-5 architecture preferably supports re-ordering of the pipeline stages

and a number of multipass rendering and processing operations, which require

feedback connections between various stages of the Cube-5 rendering pipelines 52

and the 3D memory units 24. For example, correct rendering of overlapping

5   volumetric objects preferably requires at least two passes through the Cube-5 pipeline

52, where the first pass re-samples the volumetric objects to align them with each

other and the second pass renders the volumetric objects in interleaved order. As

shown in Figure 5, a multiple volumes feedback path 66 is preferably provided,

operatively connecting the output of the compositing unit 60 to the corresponding 3D

10   memory unit 24, which allows the re-sampled volumes to be written back into the 3D

memory unit 24 after re-sampling, classification and shading. The final rendering

pass works on RGBα volumes.

Similarly, each Cube-5 rendering pipeline 52 preferably includes an image-

15   based rendering feedback path 68 connected between the warp unit 44 and the 3D

memory unit 24. The image-based rendering feedback line 68 preferably provides a

feedback path for writing the intermediate warped images to the 3D memory unit 24.

This may be particularly useful for accelerating certain image-based rendering

operations requiring multiple warp passes. The architecture of the present invention

20   further contemplates feedback connections between the 3D memory unit 24 and

various other Cube-5 rendering pipeline stages, or between the individual pipeline

stages themselves. Image rendering speed may be substantially increased by

including feedback paths which provide direct and immediate access to the

computational results of individual pipeline stages, without having to wait for the

25   results to traverse through the entire Cube-5 rendering pipeline 52.

In a preferred embodiment of the present invention, the Cube-5 system

includes connections which bypass selective stages of the rendering pipeline, that, for

example, may not be required for certain imaging operations. By bypassing these

30   unused pipeline stages, such imaging operations can be accelerated. As illustrated in

Figure 5, a texture map bypass 70 is preferably included in each Cube-5 rendering

pipeline 52. This texture map bypass connection 70 substantially speeds up mip-

20

mapping, for instance, which consists of storing multiple levels-of-detail (LOD) of the image to be processed, by bypassing the shading unit 58 and compositing unit 60 and directly presenting the results from the trilinear interpolation unit 54 and gradient estimation unit 56 to the bilinear interpolation unit 72. In this way, the architecture of the present invention can preferably be considered not only as an array of pipelines for performing volume rendering, but as a collection of hardware resources which can be selectively configured to perform a variety of imaging operations. For example, when the Cube-5 system of the present invention is performing volume rendering, essentially all of the hardware resources are required, while texture mapping generally requires only memory, some buffering and the interpolation units.

Another unique and important aspect of the present invention which will now be discussed is the ability of the Cube-5 architecture to preferably interface with at least one conventional geometry engine 76 to support mixing of geometric data and volumetric objects in a single image. This is preferably accomplished by providing at least one geometry bus, as discussed above, to interface with the geometry engine 76.

Preferably, the Cube-5 architecture of the present invention is adapted to re-use pipeline components (e.g., interpolation unit, etc.), wherever possible, to accelerate a variety of rendering algorithms using multiple configurations, in particular, rendering scenes of multiple volumetric and polygonal objects, texture mapping, and image-based rendering. Among other important advantages, reusing pipeline components reduces hardware costs. The Cube-5 architecture also supports various unique methods and algorithms for enhancing volume rendering and acceleration of other imaging operations. Some of these methods and algorithms will be discussed individually in greater detail below.

In a preferred embodiment of the Cube-5 system, formed in accordance with the present invention, volume datasets are stored in blocks, thereby taking advantage of spatial locality. Instead of linear blocking (e.g., Voxelator API), hierarchical blocks are used which are preferably stored in a distributed arrangement, skewed across multiple 3D memory units. For example, using current Mitsubishi Electric 16-

bit, 125 megahertz synchronous dynamic random access memory (SDRAM) to implement the 3D memory, each block can contain 8^3 16-bit voxels requiring 1024 bytes or two SDRAM pages.

Each block is preferably organized as a collection of 2^3-voxel miniblocks residing in the same 3D memory unit. The banks inside the SDRAM can preferably be accessed in a pipelined fashion such that the current burst transfer essentially completely hides the setup of the subsequent burst transfer. If the view-dependent processing order of the voxels in a miniblock does not coincide with their storage order, then the eight miniblock voxels are preferably reordered on the Cube-5 chip. Hence, a single copy of the volume dataset on the SDRAM is sufficient. Therefore, hierarchical blocking allows random access to miniblocks at essentially full burst mode speed, essentially full (100%) bandwidth utilization, view-independent data storage and balanced workload.

Blocking not only optimizes the memory interface, but has an additional advantage of reducing the inter-chip communication bandwidth (i.e., between Cube-5 hardware units), since only the voxels on the block perimeters need to be exchanged between neighboring chips processing neighboring blocks. While processing a b^3-voxel block in $O(b^3)$ time, only the $O(b^2)$ voxels on the block boundary need to be communicated between chips processing neighboring blocks, where b is the size of a block edge and each block has $b×b×b$ (i.e., b^3) voxels. Therefore, inter-chip communication needs $O(1/b)$ less bandwidth than with a non-blocking solution. The size of the block edge b can be in the range of about $4 \leq b \leq 64$, although a block edge size of eight (8) is preferred.

Block look-up tables (LUT) are preferably utilized to store the pointers to all blocks comprising the current volume. This approach provides an easy method to restrict the active volume while zooming into a selected region of interest of a large volume. It also allows rendering of arbitrarily shaped sub-volumes (at block-sized granularity). Additionally, scenes containing many small volumes can be rendered very efficiently, as all volumes can reside anywhere among the 3D memory units, and

only the look-up tables must be reloaded for each volume, rather than the 3D memory units.

One method of performing perspective projection and/or Level-of-Detail (LOD) relies on two-fold super-sampling in the x and y directions. Accordingly, a four-times (4×) replication of the interpolation units for trilinear interpolation, as well as the gradient estimation units for gradient computation, is preferably employed. As a result, the datapath between the SDRAM and the Cube-5 pipelines is essentially unchanged. However, the bandwidth between Cube-5 pipelines is quadrupled, as is the on-chip throughput and buffers, primarily because each sample of the normal mode is replaced by up to four samples (i.e., 2× in the x direction and 2× in the y direction).

Handling anisotropic datasets and super-sampling preferably require a modification of opacity α. The combined function is $\alpha' = 1 - (1 - \alpha)^{d/k}$, with super-sampling factor k representing the number of samples per voxel cell, and d representing the distance which a sight ray travels (i.e., the length of the path of the sight ray) through each voxel cell. Preferably, a look-up table (LUT) is employed, for fast look-up of α' during rendering.

With continued reference to Figure 5, the perspective rendering of volumetric data with close to uniform sampling of the underlying volume dataset requires re-scaling of the compositing buffer 74 with filtering between levels. Level-of-detail (LOD) perspective rendering requires re-alignment of the compositing buffer 74 between levels. Both of these processes, which incorporate global communication not available in the pipelines 52 , are preferably performed by the warp unit(s) 44. Although the compositing buffer 74 is already accessible to the warps units 44, it is preferred that a feedback line 43 be used to write the filtered values back into the compositing buffer 74.

A hardware warp unit is generally necessary to obtain final full screen images in real time (i.e., a 30 Hertz frame rate). As shown in Figure 5, the baseplane image,

generated by the compositing units 60 of the Cube-5 rendering pipelines 52, is preferably buffered in the 2D memory units 40. To lower the memory bandwidth from the 2D memory units 40 to the warp unit 44, each pixel of the baseplane image is preferably accessed only once. To perform a linear interpolation between samples of the current and the previous scanline, another FIFO unit, sized to hold at least one scanline, is required to store the previous scanline samples. The interpolation weights for each grid pixel are preferably pre-calculated on a host machine.

In order to perform the accurate mixing of volumes and geometry, for opaque geometric objects, the Z-buffer image is preferably written to the compositing buffer 60. The compositing unit 60 must perform a z-comparison prior to blending each new sample. Additionally, for translucent geometric slices, the geometry engine 76 preferably utilizes the geometry input bus (reference number 46 in Figure 4) of the present invention to insert each slab of RGBα values into the data stream so that each slab is interleaved with the volumetric data slices.

For texture mapping, Figure 6 shows, by way of example, how 32 bits of texel data are preferably stored for a 2×2 neighborhood in a miniblock of 16-bit voxels in the 3D memory unit, in accordance with the present invention. Therefore, a four-texel neighborhood of 32-bit texels is preferably read during each memory burst read. Without data duplication, the Cube-5 system preferably performs, on average, 2.25 data burst reads to access the appropriate texel neighborhood, since some texture coordinates may lie between stored miniblocks.

With reference again to Figure 5, in accordance with one form of the present invention, one way to implement image-based rendering in hardware is to utilize the memory control unit 78, preferably included in each Cube-5 pipeline 52, to read the appropriate source pixels based on the contributing region for each pipeline. The interpolation units (e.g., 54 and 72) in that pipeline 52 will then preferably perform the four-dimensional (4D) interpolations needed for light field rendering or lumigraph. As an alternative implementation, the warp unit 44 may be utilized to perform this function. The source pixels contributing to the current view are read and

assembled into the 2D memory units 40, preferably through a connection line 41, followed by the warp transformation. Preferably, four assembled source images are processed in four consecutive warp passes. The final combination of the four intermediate warped images is performed in the Cube-5 pipeline 52. As described previously above, the image-based rendering feedback line 68 provides feedback for writing the intermediate warped images to the 3D memory 24. For either approach, the 3D memory units 24 provide local storage for a large database of images.

It is to be appreciated that the apparatus of the present invention described herein above (and referred to as Cube-5) may considerably accelerate conventional volume processing methods, beyond the universal rendering already described. Additionally, the Cube-5 apparatus of the present invention may be used in conjunction with a number of unique algorithms adapted for enhancing the performance of and/or providing enhanced features for real-time volume processing, therefore making the overall Cube-5 system superior to existing volume rendering architectures, such as Cube-4. Some of these unique algorithms, including those for performing image warping, three-dimensional transformations, perspective projections, handling large volumes, high quality rendering, clipping, depth cueing, super-sampling and anisotropic datasets, are discussed in detail below.

In accordance with one form of the present invention, a method for performing image warping is presented which, among other advantages, speeds perspective warping and provides improved image quality. Image warping is preferably the final stage of the Cube-5 volume rendering pipeline. In simple terms, image warping primarily relates to the geometric transformation between two images, namely, a source image and a target image. The geometric transformation defines the relationship between source pixels and target pixels. Efficiency and high quality are equally critical issues in such applications. In the apparatus of the present invention, the warp unit preferably performs the image transformation function. Consequently, applications employing a warp unit benefit from the image warping method of the present invention.

25

Distinguished by the data flow of the transformation, image warping methods are generally classified as either forward warping or backward warping. In forward warping, the source pixels are processed in scanline order and the results are projected onto the target image. In backward warping, the target pixels in raster order are

5   inversely mapped to the source image and sampled accordingly. Most known prior art warping algorithms employ backward warping.

Compared with affine transformations (i.e., translation, rotation, scaling, shearing, etc.), a perspective transformation is considered to be more expensive and

10   challenging. For perspective projection, an expensive division is needed when calculating the sample location in the baseplane image for a pixel in the projection plane. Conventional perspective warping is typically at least three-fold slower than parallel warping, when implemented by a CPU. Accordingly, some prior art approaches have decomposed the perspective transformation into several simpler

15   transformations requiring multiple passes. One primary problem inherent in multi-pass transformation algorithms, however, is that the combination of two one-dimensional (1D) filtering operations is not as flexible as true two-dimensional (2D) filtering. Furthermore, conventional multi-pass approaches introduce additional filtering operations which degrade image quality.

20

The present invention preferably employs a unique single-pass forward warping method which can be implemented with substantially the same efficiency as affine transformations. Costly divisions, which were traditionally performed for every pixel, are reduced to only twice per scanline according to the present invention. Thus,

25   by reducing the number of division operations, the present invention provides an alternative perspective warping method which is superior to known prior art methods, at least, for example, in terms of speed and the efficient hardware implementation. A preferred method for perspective warping, in accordance with the present invention, will now be discussed.

30

Preferably, the present invention uses a scanline approach to perform perspective warping. Rather than scanning in normal raster scanline order, however,

the algorithm of the present invention is processed in a special scanline direction in the source image. As illustrated in Figures 7 and 8, this special scanline direction 92 (Figure 8) preferably has the property that parallel scanlines 84 in the source image 80 appear as parallel scanlines 86 in the target image 82, and that equi-distant sample points 88 along a source scanline 84 remain as equi-distant sample points 90 in the target scanline 86. Some advantages of this unique approach include a reduced complexity of perspective-correct image warping (i.e., by eliminating the division per pixel and replacing it with two divisions per scanline), accurate antialiasing by incorporating anisotropic filtering, correction of flaws in Gouraud shading caused by bilinear interpolation and optimization of the memory bandwidth by reading each source pixel exactly once.

The intuition of the special scanline direction is derived from projection geometry, as shown in Figure 8. Referring to Figure 8, the source image 80 is preferably placed on a three-dimensional (3D) surface and the target image 82 is placed on a screen. As in typical texture mapping, to obtain the pixel on screen, a sight ray (or rays) 94 is cast from a viewpoint (or eye point) 96 to 3D space and intersected with the screen 82 and 3D surface 80. The intersection points are the sample points 98. When the scan direction 92 in screen space is parallel to the 3D planar surface, the scanlines in both images are parallel to each other, and equi-distant sample points 98 along the scanline remain equi-distant in the 3D surface plane. This parallel-preserving (PP) scanline direction exists and is unique for a given perspective transformation. It is to be appreciated that for parallel projections, any scan direction preserves this parallelism on both images, and thus a raster scanline direction may be preferably used due to its simplicity.

Referring again to Figure 7, parallel-preserving (PP) scanlines 84 and 86 are shown in both the source 80 and target 82 images respectively. Once the parallelism property is achieved, pixel access becomes regular, and spatial coherency can be utilized in both images. Additionally, the PP scanline enables the application of a pure incremental algorithm without division to each scanline for calculating the

27

·projection of source samples 88. One division is still needed, however, for the two endpoints of every scanline due to the non-linear projection.

With continued reference to Figure 7, as the source image 80 is scanned in the PP scanline direction rather than the raster direction, sample points 90 on the target scanline 86 may not necessarily coincide with the target pixels 91. However, the sample points 90 can be aligned on the x grid lines 89 of the target image 82, thus the sample points 90 are only off the y grid lines 87 (they are equi-distant along the scanline). For a more efficient but lower quality implementation, placing the sample value in the nearest-neighbor target pixel is a reasonable approximation, as a half pixel is the maximum error. However, when higher quality is preferred, the present invention may perform pixel correction and effective antialiasing, to be described herein below.

In general, a reduction in the number of divisions from $O(n^2)$ to $O(n)$ is obtained by the algorithm of the present invention (where n is the linear resolution). For the present algorithm, preferably only two additions are needed to calculate each sample point, while conventional raster scanline algorithms generally require three additions, one division and two multiplications per pixel. A preferred method for performing forward image warping, in accordance with the present invention, is described in detail herein below.

The forward warping algorithm of the present invention is preferably performed in two stages: (1) calculating the special parallel-preserving (PP) scanline direction, and (2) forward mapping the source image to the target image along the special PP scanlines, incrementally within each scanline.

As discussed briefly above, the parallel-preserving (PP) scanline is the intersection line between the three-dimensional (3D) planar surface and the screen (i.e., target image). However, in a two-dimensional (2D) problem, the PP scanline must be calculated based on a 2D matrix. Generally, a perspective transformation can be presented as

28

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = M \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where (u, v) is the coordinate of the source pixel, (x, y) is the coordinate of the target pixel, and M is the perspective transformation matrix. The (u, v) coordinate can be expressed in terms of (x, y) as

$$(u,v) = F(x,y) = C \begin{bmatrix} ax + dy + g \\ bx + ey + h \end{bmatrix}$$

where

$$C = \frac{1}{(cx + fy + 1)}$$

5   A line in the target image can be expressed as $y = kx + B$, where slope k denotes a line direction and B denotes a line intercept. To calculate slope k for the PP scanline, two parallel lines are preferably defined having identical slope k and intercepts B of 0 and 1, represented by point pairs of $(0, 0)$, $(1, k)$ and $(0, 1)$, $(1, k + 1)$,

10   respectively. The coordinates of these points in the source image are then calculated. Since perspective transformation preserves straight lines, these two lines will remain as straight lines in the source image and their slopes can be calculated from two point pairs. Assuming that the slopes of the two mapped lines are essentially equal, an equation in k is preferably obtained. Solving this equation for k results in

$$k = -\frac{c}{f}$$

15   The corresponding slope k' in the source image is then

$$k' = \frac{bf - ec}{af - dc}$$

As can be noted from the above equation, when $k' = -\frac{c}{f}$, the denominator of the homogenous coordinates becomes a constant value of $Bf + 1$, where B is the intercept in $y = kx + B$.

The second stage of the preferred forward warping method of the present invention involves scanline processing and is illustrated in Figures 9 and 10 by way of example. Referring now to Figure 9, the preferred algorithm sweeps the scanlines 84 (e.g., scanlines $S1$ - $S4$) through the source image 80. As discussed above, the scanlines 84 have the slope k'. The samples 88 along each scanline 84 are preferably incrementally calculated. First, for each scanline 84, the projection of the endpoints from the target image onto the source image is calculated. Then, based on the number of sample points on the scanline, increments are calculated in both the x and the y directions.

Considering a traditional bilinear interpolation of samples in the source image, every sample essentially requires the contribution of four surrounding source pixels. If pixels are read every time for every sample, each pixel ought to be read four times. This leads to a memory bandwidth of four times the target image size. However, since all scanlines are in parallel, samples on neighboring scanlines usually share contributing source pixels. Consequently, in accordance with the method of the present invention, pixels that have been previously read are preferably buffered so that common pixels are read from the buffer rather than from the source image itself.

With reference to Figure 7, pixels are preferably read in a fixed pattern, called the pixel read template 100, calculated based on the Bresenham line algorithm (as appreciated by those skilled in the art). The binary digits shown at the bottom of Figure 7 represent one way of encoding the read template 100. The present invention, however, contemplates other encoding schemes, as appreciated by those skilled in the art. As illustrated in Figure 7, this code indicates the increase in the positive v direction; a "0" represents no increase and a "1" denotes an increase by one unit, while u is always increased by one unit. For the example of Figure 7, the u axis may preferably be referred to as the primary processing axis. It is preferred that the

30

template 100 always start from the left-most pixel and moves in the vertical direction (i.e., increasing v direction) so that all pixels are read and placed into the buffer for subsequent use in the sampling. It can be seen from Figure 7 that in order to provide pixels for sampling on any scanline between the two dotted lines, four pixel templates are preferably required, even though for a specific scanline, only three pixel templates might seem sufficient (e.g., only templates 2, 3 and 4 are necessary to process the current scanline $S2$). Therefore, the buffer size is preferably four scanlines.

Referring now to Figure 10A, there is illustrated the addressing of samples in the buffer. Whenever the template code value is 1, the sample decreases by one unit in the v direction. The thick zigzag line 104 represents the output scanline in the buffer. When the sample falls within the shaded region 106, in which the pixels in the buffer are sheared, care should be taken to read the correct pixels for sampling. Figure 10B illustrates a preferred procedure for bilinearly interpolating one of the samples, s, in this region.

The contents of the buffer are preferably updated based on the scanline position. For example, referring to Figure 9, templates 1, 2, 3 and 4 are preferably in the buffer when processing scanline $S1$. For scanline $S2$, the buffer preferably remains the same. For scanline $S3$, template 5 is preferably read into the buffer and template 1 is discarded. For scanline $S4$, template 6 preferably replaces template 2, and so on.

As mentioned above, one of the features of the unique forward image warping method of the present invention is the correction of flaws in Gouraud shading. Gouraud shading is a popular intensity interpolation algorithm used to shade the surfaces of geometric objects. Given color only at the vertices, Gouraud shading bilinearly interpolates the intensities for the entire rasterization of a geometry in a raster scanline order. The flaws of the Gouraud shading approach are known in the art and have been the subject of such articles as, for example, Digital Image Warping, by G. Wolberg, *IEEE Computer Society Press*, 1990.

31

One of the problems associated with the Gouraud approach is that diagonal lines (as an example) are not linearly mapped for perspective projections. When a diagonal line is perspectively projected onto a screen in 3D screen space, Gouraud shading converts this diagonal line into a curve, which violates the property of
5   preserving lines in perspective transformation.

The image warping method of the present invention corrects the perspective distortion in Gouraud shading. The perspective distortion is present because the linear interpolation along a raster in screen space is generally non-linear when transformed
10   into geometrical coordinates. Using the special scan direction of the present invention, however, linearity is preserved by the mapping. Thus, interpolation is linear in both image and geometrical space, thereby fixing the distortion of Gouraud shading. It is to be appreciated that interpolation along the edges is still non-linear, and therefore the scanline endpoints must be transformed into geometrical space for
15   correct interpolation.

The forward mapping algorithm of the present invention, with nearest-neighbor approximation, preferably generates a target image that is essentially indistinguishable from an image generated using traditional methods. However, when
20   a higher image quality is desired, the method of the present invention can preferably calculate the pixel value at exact grid points. A simple target pixel correction scheme may preferably be introduced to perform this correction.

With reference now to Figure 11, assuming the sample points 90 in the target image 82 are aligned on integer x coordinates, in order to obtain the pixel value at the
25   exact pixel grid locations 91, a linear interpolation of the two samples immediately above and below each pixel is preferably performed. Performing this linear interpolation simply as a second pass may increase the cost, since the samples must be read over again. Instead, as each sample is generated, a preferred method of the
30   present invention spreads the contribution of each sample to the corresponding upper and lower pixels with no intermediate buffering.

32

As illustrated by the example of Figure 11, samples 112 located on the thicker inclined scanline 108 contribute to the shaded pixels neighboring them (lighter shading above the scanline, darker shading below the scanline). The arrows indicate that each sample 112 preferably contributes to two pixels. It is preferred that a pixel not be written out until both contributions are collected. Thus, a one scanline buffer is preferably included for storing the intermediate pixel values.

To write out pixels correctly and efficiently, a pixel write pattern, called a pixel write template 110, is preferably pre-calculated. Unlike the pixel read template (e.g., reference number 100 in Figure 9), the pixel write template 110 is preferably calculated by truncating the y coordinate value of samples along a scanline. The template 110 is preferably encoded as a series of integer y steps and fractional distances dy from the true scanline 86. The weights used for the final linear interpolation are dy and $1 - dy$ for the upper and lower pixels, respectively. Since all scanlines are preferably one unit apart in the vertical direction (i.e., y direction), the template is calculated only once per projection.

The forward image warping method of the present invention can further improve on image quality by antialiasing. Using the parallel-preserving (PP) scanline, a higher quality, less expensive method of antialiasing may be achieved.

Referring again to Figure 7, the sample points on the upper scanlines of the source image are sparser than on the lower scanlines, resulting in a transition from under-sampling to normal sampling. Thus, an appropriate resampling filter may preferably be used to avoid aliasing on the upper scanlines. Isotropic filtering results in clearly incorrect and blurry images. The need for anisotropic filters has been addressed in such articles as Survey of Texture Mapping, by P. S. Heckbert, *IEEE Computer Graphics and Applications*, 6(11):56-67, November 1986, and more recently in Texram: Smart Memory for Texturing, by A. Schilling, et al., *IEEE Computer Graphics and Applications*, 16(3):32-41, May 1996.

It is known by those skilled in the art, that each filter is defined by its footprint and profile. Taking a target sample as a circle, its projection in the source image is its footprint. As illustrated in Figure 12, this footprint 114 should generally be neither circular (i.e., isotropic) nor square-shaped (i.e., as in mip-mapping), but conic in

5   shape. The profile of the filter decides the weights of the contributing pixels within the footprint. Although a sinc filter is optimal, a gaussian filter is easier to implement and is preferred because of its finite footprint and good low-pass characteristics. The perspective warping algorithm of the present invention offers more accuracy in calculating the anisotropic footprint, producing higher image quality at a lower cost.

10

Using conventional methods for calculating the anisotropic footprint, the main axes of the ellipse must be calculated for every pixel. Although approximations have been proposed, this remains an expensive computation, and no known incremental method is available. To obtain the major axes of the ellipse using these prior art

15   methods, the Jacobian must be calculated. Using the image warping method of the present invention, however, calculation of the Jacobian may be eliminated.

In order to gain insight into a preferred method for calculating the anisotropic footprint in accordance with the present invention, the properties of the Jacobian will

20   first be analyzed. The generalized backward mapping from an xy target image into a uv source image was previously defined above as

$$\begin{bmatrix} u \\ v \end{bmatrix} = F(x,y) = C \begin{bmatrix} ax + dy + g \\ bx + ey + h \end{bmatrix}$$

where

$$C = \frac{1}{(cx + fy + 1)}$$

The Jacobian J for the generalized transformation is a non-linear function of x and y,

25     $$J = C^2 \begin{bmatrix} y(af - cd) + a - gc & x(af - cd) - d + gf \\ y(bf - ce) + b - hc & x(bf - ce) - e + hf \end{bmatrix}$$

In conventional antialiasing approaches, the Jacobian is used to determine the footprint of each pixel in the source image and is necessary for anisotropic filtering. The differences between screen pixels in xy raster space are projected into the source image by computing the directional derivatives in the [1, 0] and [0, 1] directions. These derivatives in source image space are called $r1$ and $r2$, and are defined as

$$r_1 = J \begin{bmatrix} 1 \\ 0 \end{bmatrix} = C^2 \begin{bmatrix} y(af - cd) + a - gc \\ y(bf - ce) + b - hc \end{bmatrix}$$

and

$$r_2 = J \begin{bmatrix} 0 \\ 1 \end{bmatrix} = C^2 \begin{bmatrix} x(af - cd) - d + gf \\ x(bf - ce) - e + hf \end{bmatrix}$$

These vectors, r_1 and r_2, define the bounding box of an ellipse that approximates the footprint 114. Typically, these vectors 116 and 118 are calculated for every pixel, when needed, for conventional methods of anisotropic filtering (e.g., elliptical weighted average (EWA), footprint assembly). This requires one more division per pixel for calculating C. In accordance with the present invention, a more accurate method for determining the footprint is presented, as described herein below.

Because the Jacobian is a linear approximation of the non-linear mapping, it is more accurate, and therefore preferable, to compute the footprint by taking the distances to neighboring samples in source image space. Since the projections of neighboring samples are already computed, this method of the present invention requires no additional division.

The parallel-preserving (PP) scan direction provides for greater coherency and no division to compute the Jacobian. For each pixel in the PP scanning order, the footprint is preferably defined by r_1' and r_2'. The directional derivative r_1' in direction [1, k] along the PP scanline is

$$r_1' = \nabla_{[1,k]}F = J\begin{bmatrix} 1 \\ k \end{bmatrix} = C^2\begin{bmatrix} af - cd \\ bf - ce \end{bmatrix}$$

and since $y = kx + B$, $C = \dfrac{1}{(Bf+1)}$ is constant for every PP scanline, and thus r_1' is constant for every PP scanline. The method of the present invention exploits this fact in order to preferably increment the source image coordinates along a scanline, with no divisions. The value of the directional derivative r_2' in the y direction $[0, 1]$ is

$$r_2' = \nabla_{[0,1]}F = r_2$$

It is to be appreciated that r_2' varies linearly along the scanline since it is a function of x, and thus it can be incremented along the scanline.. The special scan direction makes it possible to compute the source image coordinates and pixel footprints simply and efficiently.

After efficiently computing all the footprint and source pixel coordinate information, correct anisotropic filtering can be performed using a standard method known by those skilled in the art, such as, for example, Greene and Heckbert's elliptical weighted average (EWA) or Shilling et al.'s footprint assembly. These conventional algorithms are described, for example, in the text Creating Raster Omnimax Images from Multiple Perspective Views Using the Elliptical Weighted Average Filter, by N. Greene and P. S. Heckbert, *IEEE Computer Graphics and Applications*, 6(6):21-27, June 1986. However, these conventional filtering approaches are not preferred since, as pointed out previously, even the elliptical footprint approximation is inaccurate. Furthermore, such prior art methods result in redundant sampling (i.e., accessing each source pixel multiple times). For instance, for a circular filter region with a footprint radius of 1.0 source pixel, each source pixel is sampled an average of π times. By using the forward mapping technique of the present invention, redundant memory access can be essentially eliminated, thus lowering the memory bandwidth by a factor of π. Preferably, the present invention provides a forward mapping technique in which all source pixels are read once in

pixel read template order and subsequently splatted onto the target image with a filter kernel.

As illustrated in Figure 13, each source pixel 124 has a Δx 120 and a Δy 122 relative to each of its nearest-neighbor target samples 126. The Δx can be preferably computed incrementally since all samples along a scanline are equi-distant. The special scan direction essentially guarantees that the Δy is constant along each scanline. Although the raster grid locations deviate from the true scanline 128, the actual distances can be estimated preferably by adding a small correction which may be stored in the pixel read template 130 and is preferably uniform among scanlines. The filter kernel is preferably pre-computed once and stored in a lookup table (LUT). Subsequently, the contribution of each source pixel 124 is preferably indexed by its Δx and Δy into the lookup table (LUT) for the four (or more) nearest-neighbor target samples 126. The number of target samples 126 depends upon the footprint of the filter used, and it may preferably vary from four to 16 samples. Using this method, each source pixel 124 is preferably read exactly once from memory, then four (or more) times modulated by a lookup table entry and accumulated in the target pixel. In this manner, the final pixel value is the weighted average of the nearby source pixels 124. This weighted average requires a division by the sum of the filter weights to normalize each final pixel intensity.

In addition to image warping, which can be broadly defined as a geometric transformation between two images (e.g., a source image and a target image), three-dimensional (3D) volume transformation plays a key role in volume rendering, volume modeling and registration of multiple volumes. Among all affine transformations, rotation generally consumes the most computation time and is considered the most complicated. Accordingly, in providing a universal 3D rendering architecture in accordance with the present invention, several unique methods for performing arbitrary 3D volume rotation are presented, as described in detail herein below. Although the universal 3D rendering hardware of the present invention may be used without the 3D volume rotation methods described herein, these methods, or algorithms, are preferably implemented in conjunction with the apparatus of the

present invention to provide enhanced speed and features and are adapted to most efficiently utilize the apparatus of the present invention.

Prior to describing the unique methods for performing 3D volume rotation, it is important to first provide some basic definitions of the terms used. As appreciated by those skilled in the art, relative to the rows and columns of an image, a *beam* in a volume may be defined as a row of voxels along one major coordinate axis (e.g., an *x*-beam is a row of voxels in the *x* direction). A *slice* of a volume is a plane of voxels which is perpendicular to a major axis (e.g., an *x*-slice is defined as a plane perpendicular to the *x* axis).

Prior art methods for performing volume transformations typically utilize multiple-pass algorithms, which are usually direct extensions of the multiple-pass algorithms used for image transformations. Various methods for performing 3D rotation have been proposed, generally involving a decomposition of the 3D transformation into multiple two-dimensional (2D) or one-dimensional (1D) transformations. These prior art methods have been the subject of articles, including Volume Rendering, by R. A. Drebin et al., *Computer Graphics (SIGGRAPH '88 Proceedings)*, Vol. 22, pp 65-74, August 1988, Three-Pass Affine Transformations for Volume Rendering, by P. Hanrahan, *Computer Graphics (San Diego Workshop on Volume Visualization)*, Vol. 24, pp 71-78, November 1990 and Fast Rotation of Volume Data on Parallel Architectures, by P. Schroder and J. B. Salem, *Visualization '91*, pp. 50-57, 1991, all of which are incorporated herein by reference. However, these known 3D transformation methods typically result in a lower quality rotation and/or slower processing speed.

One of the properties which make three-dimensional (3D) rotation so difficult is that 3D rotations inherently require global communication and could cause memory contention while writing data back to the distributed memory modules. However, as shear transformation capitalizes on nearest neighbor connections, it lends itself to an extremely feasible multi-pipelined hardware implementation, as provided by the unique architecture of the present invention. The present invention further provides

novel methods for performing arbitrary 3D rotation, essentially by decomposing the 3D rotations into sequences of different types of shear transformations.

Using a conventional decomposition approach, since a 2D rotation can be decomposed into three one-dimensional (1D) shears, a direct extension to 3D rotation would require nine 1D shears. However, in accordance with the present invention, four preferred methods of shear decomposition of an arbitrary 3D volume rotation are presented, as described in detail herein below. These methods include a four-pass 2D slice shear, a four-pass 2D beam shear, a three-pass beam-slice shear and a two-pass 3D beam shear decomposition. By not introducing a scale operation, the algorithms of the present invention avoid complications in sampling, filtering and the associated image degradations.

It is to be appreciated by one skilled in the art that a 3D rotation matrix can be expressed as the concatenation of three major axis rotations, $R_x(\phi)$, $R_y(\theta)$, $R_z(\alpha)$, where

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 0 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The order in which this concatenation is performed results in different 3D rotation matrices. There are six permutations of the 3D rotation matrix in total. By way of illustration, the underlying 3D rotation matrix was chosen as $R_{3D} = R_x(\phi)R_y(\theta)R_z(\alpha)$, where

$$R_{3D} = \begin{bmatrix} \cos\theta\cos\alpha & \cos\theta\sin\alpha & -\sin\theta \\ \sin\phi\sin\theta\cos\alpha - \cos\phi\sin\alpha & \sin\phi\sin\theta\sin\alpha + \cos\phi\cos\alpha & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\alpha + \sin\phi\sin\alpha & \cos\phi\sin\theta\sin\alpha - \sin\phi\cos\alpha & \cos\phi\cos\theta \end{bmatrix}$$

The above 3D rotation matrix (R_{3D}) is used for all the decompositions which follow. One of the primary differences between the unique methods of the present invention and other conventional approaches is that in the present invention, the decomposition is applied directly to a 3D rotation matrix, rather than to multiple 2D rotation sequences, to obtain shear sequences. It is to be appreciated that, for any of the shear operations performed in accordance with the present invention, barrel shifters may be used as a preferred hardware implementation, although other means, such as logarithmic shifters or the like, are similarly contemplated.

As shown in Figure 14, a method for performing two-dimensional (2D) slice shear rotation, in accordance with one embodiment of the present invention, preferably involves a decomposition of the 3D rotation into a sequence of 2D slice shears. In a 2D slice shear, a volume slice (i.e., a plane of voxels along a major projection axis and parallel to any two axes) is merely shifted within its plane. A slice may be arbitrarily taken along any major projection axis. For example, Figure 14 illustrates a y-slice shear. A 2D y-slice shear is preferably expressed as:

$$x = x + a \cdot y$$
$$z = z + b \cdot y$$

A 2D y-slice shear may preferably be written as $S(xz, y, (a, b))$, interpreted as a shear along the y axis by an amount a 132 in the x-direction and an amount b 134 in the z-direction. Although both a and b are preferably constants, it is further contemplated that a and b can represent functions as well. A 2D x-slice shear, $S(yz, x, (c, d))$, and a 2D z-slice shear, $S(xy, z, (e, f))$, are similarly defined. With reference to

Figure 14, the volume represented by the solid lines 136 is the shear result of the volume defined by the dotted lines 138.

Intuitively, consecutive shears along the same axis produce a conforming shear. For example:

$$S(xz,y,(a,b)) \cdot S(xz,y,(a',b'))$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ a & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ a' & 1 & b' \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ a+a' & 1 & b+b' \\ 0 & 0 & 1 \end{bmatrix}$$

In order to build the general 3D matrix from 2D shear matrices, shear products may be restricted to products of different shears: *S(yz, x, (c, d))*, *S(xz, y, (a, b))* and *S(xy, z, (e, f))*. However, the product matrix of these three shear matrices will still not be in the general form due to a constant 1 in the present matrix. Accordingly, another shear matrix is preferably concatenated, where this final shear is the same slice shear as the first one. This results in the following six permutations of shear sequences:

$$S(xz,y,(a,b)) \cdot S(xy,z,(e,f)) \cdot S(yz,x,(c,d)) \cdot S(xz,y,(g,h))$$

$$S(xz,y,(a,b)) \cdot S(yz,x,(c,d)) \cdot S(xy,z,(e,f)) \cdot S(xz,y,(g,h))$$

$$S(xy,z,(e,f)) \cdot S(xz,y,(a,b)) \cdot S(yz,x,(c,d)) \cdot S(xy,z,(i,j))$$

$$S(xy,z,(e,f)) \cdot S(yz,x,(c,d)) \cdot S(xz,y,(a,b)) \cdot S(xy,z,(i,j))$$

$$S(yz,x,(c,d)) \cdot S(xz,y,(a,b)) \cdot S(xy,z,(e,f)) \cdot S(yz,x,(m,n))$$

$$S(yz,x,(c,d)) \cdot S(xy,z,(e,f)) \cdot S(xz,y,(a,b)) \cdot S(yz,x,(m,n))$$

For each of the shear sequences, the product matrix of the consecutive shear matrices is preferably computed and set equal to the underlying 3D rotation matrix. For example, for the first shear sequence given above (i.e., *S(xz, y, (a, b)) S(xy, z, (e, f)) S(yz, x, (c, d)) S(xz, y, (g, h))*):

41

$$R_{3D} = R_x(\phi)\,R_y(\theta)\,R_z(\alpha) =$$

$$S(xz,y,(a,b)) \cdot S(xy,z,(e,f)) \cdot$$

$$S(yz,x,(c,d)) \cdot S(xz,y,(g,h))$$

The above matrix equation implies nine trigonometric equations with eight variables, namely, a, b, c, d, e, f, g and h. In solving these nine equations for the eight variables, a - h, the following results are obtained:

$$a = \frac{\sin\theta\sin\alpha(\cos\theta - \cos\phi) + \sin\phi(\cos\alpha - \cos\theta)}{(\cos\theta)^2 \sin\alpha\sin\phi}$$

$$b = -\frac{\cos\phi\cos\theta - 1}{\sin\phi\cos\theta}$$

$$c = \cos\theta\sin\alpha$$

$$d = \frac{-\sin\phi\sin\theta\cos\alpha + \cos\phi\sin\alpha - \cos\theta\sin\alpha}{\sin\phi\cos\theta}$$

$$e = \frac{\sin\phi\cos\theta + \cos\phi\sin\theta\sin\alpha - \sin\phi\cos\alpha}{\cos\theta\sin\alpha}$$

$$f = -\sin\phi\cos\theta$$

$$g = \frac{\cos\theta\cos\alpha - 1}{\cos\theta\sin\alpha}$$

$$h = \frac{\cos\theta\cos\alpha - 1}{\cos\theta\sin\alpha}$$

In a similar manner, the shear matrices for the remaining five slice shear sequences given above may be obtained. In fact, the slice shear sequence with the solution given above has the simplest expression and is preferably termed the *dominant* sequence.

Referring now to Figure 15, a method for performing three-dimensional (3D) rotation by a two-dimensional (2D) beam shear decomposition will now be described. First, a beam shear may be defined as a beam that is merely shifted in its major

direction without any change of the other two coordinates. For example, a 2D x-beam shear is preferably expressed as:

$$x = x + a \cdot y + b \cdot z$$

A 2D x-beam shear may preferably be written as $S(x, yz, (c, d))$, interpreted as a shear along the x axis by an amount a in the x-direction and an amount b in the z-direction. A 2D y-beam shear, $S(y, xz, (a, b))$, and a 2D z-beam shear, $S(z, xy, (e, f))$, are similarly defined. Figure 15 illustrates an x-beam shear, wherein the volume represented by the dotted lines 146 is sheared to the volume position represented by the solid lines 144.

A two-dimensional (2D) beam shear is advantageous over a 2D slice shear, and is therefore preferred, since a beam is shifted without changing the other two coordinates. Thus, the resampling for each pass of the 2D beam shear approach is simpler, as only a linear interpolation is required. In contrast, a 2D slice shear approach requires a bilinear interpolation which is more complex.

Similar to the 2D slice shear decomposition, in order to build the general 3D matrix from 2D beam shear matrix decompositions, shear products may be restricted to products of different shears: $S(x, yz, (c, d))$, $S(y, xz, (a, b))$, $S(z, xy, (e, f))$. However, the product matrix of these three shear matrices will still not be in the general form due to a constant 1 in the matrix. Accordingly, as in the slice shear method, another shear matrix is preferably concatenated, where this final shear is the same beam shear as the first one. This results in the following six permutations of shear sequences:

$$S(y, xz, (a, b)) \, S(z, xy, (e, f)) \, S(x, yz, (c, d)) \, S(y, xz, (g, h))$$

$$S(y, xz, (a, b)) \, S(x, yz, (c, d)) \, S(z, xy, (e, f)) \, S(y, xz, (g, h))$$

$$S(z, xy, (e, f)) \, S(y, xz, (a, b)) \, S(x, yz, (c, d)) \, S(z, xy, (i, j))$$

$$S(z, xy, (e, f)) \, S(x, yz, (c, d)) \, S(y, xz, (a, b)) \, S(z, xy, (i, j))$$

$$S(x, yz, (c, d)) \, S(y, xz, (a, b)) \, S(z, xy, (e, f)) \, S(x, yz, (m, n))$$

$$S(x, yz, (c, d)) \, S(z, xy, (e, f)) \, S(y, xz, (a, b)) \, S(x, yz, (m, n))$$

For each of the above shear sequences, the product matrix of the consecutive shear matrices is preferably computed and set equal to the underlying 3D rotation matrix. For example, for the first shear sequence given above (i.e., $S(y, xz, (a, b))$ $S(z, xy, (e, f))$ $S(x, yz, (c, d))$ $S(y, xz, (g, h))$):

$$R_{3D} = R_x(\phi) R_y(\theta) R_z(\alpha) =$$

$$S(y, xz, (a, b)) \cdot S(z, xy, (e, f))$$

$$\cdot S(x, yz(c, d)) \cdot S(y, xz, (g, h))$$

The above matrix equation implies nine trigonometric equations with eight variables, namely, a, b, c, d, e, f, g and h. In solving these nine equations for the eight variables, $a - h$, the following results are obtained:

$$a = \frac{\sin\theta\sin\alpha(\cos\phi - \cos\theta) + \sin\phi(\cos\theta - \cos\alpha)}{\sin\phi\sin\alpha(\cos\theta)^2}$$

$$b = -\frac{\cos\phi\cos\theta - 1}{\sin\phi\cos\theta}$$

$$c = -\cos\theta\sin\alpha$$

$$d = \frac{\cos\theta\sin\alpha + \sin\phi\sin\theta\cos\alpha - \cos\phi\sin\alpha}{\sin\phi\cos\theta}$$

$$e = -\frac{\cos\phi\sin\theta\sin\alpha - \sin\phi\cos\alpha + \sin\phi\cos\theta}{\cos\theta\sin\alpha}$$

$$f = \sin\phi\cos\theta$$

$$g = -\frac{\cos\theta\cos\alpha - 1}{\cos\theta\sin\alpha}$$

$$h = \frac{\sin\phi\sin\theta(\cos\theta - \cos\alpha) + \sin\alpha(\cos\phi - \cos\theta)}{\sin\phi\sin\alpha(\cos\theta)^2}$$

In a similar manner, the shear matrices for the remaining five beam shear sequences given above may be obtained. The beam shear sequence with the solution given above is preferably termed the *dominant* sequence.

44

With reference now to Figure 16, a method for performing three-dimensional (3D) rotation by two-dimensional (2D) beam-slice shear decomposition in accordance with the present invention will be described. A 2D beam-slice shear may preferably be defined as a beam that is shifted within a plane. For example, a 2D x-beam-y-slice

5   shear is preferably expressed as:

$$x = x + a \cdot y + g \cdot z$$
$$z = z + b \cdot y$$

A 2D x-beam-y-slice shear may preferably be written as $S((x, yz, (a, g)), (z, y, b))$, interpreted as a shear along the x axis by an amount a in the y-direction and an amount g in the z-direction, combined with a shear along the z axis by an amount b in the y-direction, where a, g and b are preferably constants. In essence, a beam-slice

10   shear is a combination of a beam shear and a slice shear. Figure 16 illustrates an x-beam-y-slice shear, $S((x, yz, (a, g)), (z, y, b))$, wherein the volume represented by the dotted lines 156 is sheared to the volume position represented by the solid lines 154.

To build the general 3D matrix from a 2D shear matrix decomposition, shear

15   products may be restricted to products of different shears: y-beam-x-slice shear $S((y, xz, (c, h)), (z, x, d))$, x-beam-y-slice shear $S((x, yz, (a, g)), (z, y, b))$, and y-beam shear $S(y, xz, (I, f))$. As in the case of the slice shear and beam shear approaches, it is to be appreciated that there are also six permutations of beam-slice shear sequences.

20   For each of the shear sequences, the product matrix of the consecutive shear matrices is preferably computed and set equal to the underlying 3D rotation matrix. For example, for the first beam-slice shear sequence given above (i.e., $S((y, xz, (c, h)), (z, x, d)) S((x, yz, (a, g)), (z, y, b)) S(y, xz, (I, f)))$:

$$R_{3D} = R_x(\phi) R_y(\theta) R_z(\alpha) =$$

$$S((y, xz, (c, h)), (z, x, d)) \cdot S(x, yz, (a, g)), (z, y, b))$$

$$\cdot S(y, xz, (i, f))$$

45

The above matrix equation implies nine trigonometric equations with eight variables, namely, a, b, c, d, f, g, h and I. In solving these nine equations for the eight variables, the following results are obtained:

$$a = \sin\phi\sin\theta\cos\alpha - \cos\phi\sin\alpha$$

$$b = \sin\phi\cos\theta$$

$$c = \frac{\sin\phi(\cos\theta - \cos\alpha) + \sin\theta\sin\alpha(\cos\phi - \cos\theta)}{\sin\phi(\cos\theta)^2\sin\alpha}$$

$$d = \frac{\sin\phi\cos\alpha - \cos\phi\sin\theta\sin\alpha - \sin\phi\cos\theta}{\cos\theta\sin\alpha}$$

$$f = \frac{\sin\phi\sin\theta(\cos\theta - \cos\alpha) + \sin\alpha(\cos\phi - \cos\theta)}{\sin\phi(\cos\theta)^2\sin\alpha}$$

$$g = \frac{\cos\theta\sin\alpha + \sin\phi\sin\theta\cos\alpha - \cos\phi\sin\alpha}{\sin\phi\cos\theta}$$

$$h = \frac{\cos\phi\cos\theta - 1}{\sin\phi\cos\theta}$$

$$i = -\frac{\cos\theta\cos\alpha - 1}{\cos\theta\sin\alpha}$$

It is to be appreciated that the shear matrices for the remaining five shear

5   sequences may be obtained in a similar manner.

Figure 17 illustrates a fourth method for performing an arbitrary three-dimensional (3D) rotation using 3D beam shear decompositions, according to the present invention. By further examination of the product matrix of the consecutive

10   shear matrices used in the beam-slice shear decomposition method described above (i.e., $S((y, xz, (c, h)), (z, x, d)) \cdot S((x, yz, (a, g)), (z, y, b)) \cdot S(y, xz, (I, f)))$, the first pair and the last pair of 2D shears can be merged since there is a common beam in each pair. For example, x beam is a common beam of the y-slice and z-slice shears of the first pair. Therefore, the number of shears can be reduced to two by introducing a

15   new definition of a 3D beam shear.

Figure 17 illustrates a 3D x-beam shear, which is equal to the concatenation of two consecutive 2D slice shears $S(xz, y, (a, b))$ $S(xy, z, (e, f))$. It is to be appreciated that there are two other 3D beam shears, namely, a 3D z-beam shear, represented as $S(yz, x, (c, d))$ $S(xz, y, (a, b))$, and a 3D y-beam shear, represented as $S(yz, x, (c, d))$ $S(xy, z, (e, f))$. Every 3D beam shear preferably involves only one major beam. With reference to Figure 17, the marked x beam 158 (dark shaded beam) is preferably translated to a new 3D location following the arrows. The lighter shaded beam 158' indicates the intermediate position if the shear decomposition is interpreted as two consecutive 2D slice shears.

The three 3D beam shears may preferably be denoted as SH_{3D_x}, SH_{3D_y} and SH_{3D_z}. Now, using the method of the present invention described herein, an arbitrary 3D rotation can be decomposed into only two consecutive 3D beam shears. The dominant decomposition sequence may be obtained directly from the 2D slice shear sequence as:

$$R_{3D} = SH_{3D_x} \cdot SH_{3D_z}$$

where

$$SH_{3D_x} = S(xz, y, (a, b)) \cdot S(xy, z, (e, f))$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ a+be & 1+bf & b \\ e & f & 1 \end{bmatrix}$$

$$SH_{3D_z} = S(yz, x, (c, d)) \cdot S(xz, y, (g, h))$$

$$= \begin{bmatrix} 1+cg & c & d+ch \\ g & 1 & h \\ 0 & 0 & 1 \end{bmatrix}$$

Using the 3D beam shear decomposition approach of the present invention described herein, an arbitrary 3D rotation preferably involves only two major beam

transformations, whereas conventional decomposition approaches require three (e.g., Hanrahan's decomposition). In accordance with the 3D beam shear method of the present invention, the first pass involves only x beams and the second pass involves only z beams. By the end of the first shear pass, all voxels of a beam preferably have

5   the same offsets. As there are N^2 beams for an N^3 volume, there are only N^2 different offset values. Accordingly, the offset values for N^2 beams can be stored at the end of the first pass, while storing the voxels to their nearest neighbor integral positions.

When multiple pass algorithms are used, the resampling techniques chosen are

10   key to achieving high quality. Intuitively, resampling is necessary for each pass because a continuous shear transformation may move voxels off the grid points. One problem inherent with multiple resampling, however, is the quick degradation of the volume quality if consecutive rotations are applied to a volume. It is therefore desirable to sample the volume only once.

15

Accordingly, a preferred method of the present invention achieves one pass resampling of a volume. In essence, the method of the present invention involves precomputing a sampled volume and then using only zero-order (i.e., nearest neighbor) interpolation in each shear pass, thereby distinguishing from known prior

20   art methods which require global communication (e.g., Wittenbrink and Somani's permutation warping).

Given an original volume (source volume) and the desired rotated volume (target volume), the method of the present invention preferably first builds up a one-

25   to-one correspondence between a source voxel and a target voxel. This one-to-one mapping is guaranteed by the multi-pass shear decomposition of the present invention because each shear is a one-to-one transformation using zero-order interpolation. The concatenation of a sequence of one-to-one mapping remains one-to-one. Once this one-to-one correspondence is built up, the method of the present invention preferably

30   calculates for each source voxel its corresponding target voxel and stores it in the source voxel position. During this procedure, no global communication is required; the resampling is performed by interpolation on the local voxels. The sampling

position of each target voxel is preferably computed using a backward transformation of rotation.

After obtaining the values for all target voxels, the method of the present invention preferably shuffles them to their destinations in target volume. Intuitively, this would involve global communication. However, global communication is expensive to perform for parallel implementation. Therefore, the method according to present invention preferably uses multiple shears with a nearest neighbor placement scheme to achieve this voxel shuffling. Since shear is a regular, non-conflict transformation, each pass can be performed more efficiently than if global communication was utilized. Using the 3D beam shear decomposition method of the present invention described herein, only a minimum of two passes of regular local communication are necessary to achieve virtually the same effect as global communication.

It is to be appreciated that care should be taken to avoid the overlapping of beams in 3D beam shear. Consider, for example, the 3D x beam shear equation given above. While each x beam is preserved (i.e., an x beam remains rigid after a 3D x beam shear), several x beams may overlap with each other. To maintain the required one-to-one mapping, recall that a 3D beam shear is the concatenation of two 2D slice shears, as discussed above. A 2D slice shear maintains one-to-one mapping when using zero-order interpolation. Therefore, as a solution, the method of the present invention preferably calculates the destination coordinates using the same order as that of two consecutive 2D slice shears, but communication is preferably only performed once. For a 3D x beam shear, while the x coordinate is calculated directly using the 3D shear matrix (described above), the y and z coordinates of each beam are preferably calculated as

$$z' = round(z + b \cdot y)$$

$$y' = round(y + f \cdot z')$$

49

where *round(w)* is a function of rounding *w* to the nearest integer. Coordinates (y', z') determine the integral coordinates of the whole beam for the nearest neighbor storage. In this manner, no overlap occurs.

5   In accordance with another form of the present invention, several unique methods for performing enhanced volume processing will be discussed in detail herein below.

           Perspective projections present inherent challenges, particularly when
10 performing ray casting. For parallel projections, sight rays that are cast through a volume dataset maintain a constant sampling rate on the underlying volume data. It is straightforward to set this sampling rate to create an output image of the required quality. For perspective projections, however, the rays do not maintain such a continuous and uniform sampling rate. Instead, the rays diverge as they traverse the
15 volume from front to back. This creates an uneven sampling of the underlying volume, as shown in Figures 18A and 18B.

           Referring now to Figures 18A and 18B, conventional ray casting algorithms generally handle ray divergence from perspective projections by one of two methods.
20 The first method is undersampling (Figure 18A), in which rays 160 are cast, from a predefined viewpoint 162, so that the sampling rate at the front of the volume 164 is appropriate for the desired image quality. However, because of the perspective ray divergence, the underlying volume dataset is undersampled. This may result in severe aliasing by creating "holes" in the rear of the volume 166 where regions of voxels
25 remain unsampled. The second method is oversampling (Figure 18B), in which rays 160 are cast from a predefined viewpoint 162 so that the sampling rate at the rear of the volume dataset 166 is appropriate for the desired image quality. This approach avoids the aliasing of the first method; however, the volume may be radically oversampled in the front 164. The inefficient oversampling in the front of the volume
30 164 dramatically increases the runtime of this method. The rays 160 can be cast with a sampling rate between undersampling and oversampling. This results in a tradeoff between the image quality of oversampling and the rendering speed of undersampling.

50

Many prior imaging architectures do not even attempt to perform perspective projections. Other architectures have dealt with perspective projections by casting diverging sight rays from a predefined viewpoint, which produce images with temporal aliasing and either do not achieve true real-time frame rates (i.e., 30 Hertz) or are much more complex than the slice-order method of the present invention.

A ray-splitting method applies the concept of adaptive super-sampling in order to maintain a uniform ray density. In this approach, a ray is split into two child rays when neighboring rays diverge beyond some predetermined threshold. Recently, a method was proposed which divides the viewing frustum into regions based on distance from the viewpoint, such that the ray density in each region is near the underlying volume resolution. Afterwards, such method projects each region onto sub-images and composites them into the frame buffer using texture mapping hardware. In effect, the technique casts continuous rays through a region, then at specified boundaries, splits them into a new set of continuous rays. This, however, creates a potential undesired discontinuity between regions.

A method for performing perspective projections of uniform regular datasets, termed *ER-Perspective* (exponential regions perspective), in accordance with one form of the present invention, preferably adaptively samples the underlying volume, whereby the above-described problems, inherent in conventional volume rendering systems and methods, are essentially eliminated. The ER-Perspective algorithm combines the desirable properties of both undersampling and oversampling, providing extremely good anti-aliasing properties associated with oversampling methods, while providing runtimes on the order of undersampling methods. Furthermore, this algorithm preferably creates at least one sample for every visible voxel in the volume dataset. ER-Perspective gains a runtime advantage over previous work by utilizing slice-order voxel access, while maintaining equal or better image quality in comparison to known perspective projection methods.

Figure 19 is a 2D top view illustration of the ER-Perspective algorithm, in accordance with the present invention. As shown in Figure 19, the ER-Perspective

algorithm preferably works by dividing a view frustum 168 into a plurality of regions based on exponentially increasing distances along a major projection axis (e.g., z-axis) from a predefined viewpoint 172. Preferably, continuous sight rays 174 are cast from the viewpoint 172 from back-to-front (or front-to-back) through the volume dataset and the rays 174 are merged (or split) once they become too close (or too far) from each other. Since the operation of the ER-Perspective algorithm is similar for back-to-front compared with front-to-back ray casting, the remaining discussion of the ER-Perspective algorithm will be limited to the more intuitive case of back-to-front ray casting with merging. The differences are pointed out where they are significant.

The ER-Perspective algorithm preferably uses region boundaries 170, which define the exponential regions, to mark the locations where the sight rays 174 are preferably merged. By defining the regions and merging all rays 174 at the boundaries 170, the algorithm provides a regular pattern of ray merging that is dependent on the global geometry rather than local neighborhood conditions. Figure 20A more clearly illustrates the merging of sight rays at region boundaries 170 for contribution to baseplane pixel B, in particular. With reference to Figure 20A, an odd number of rays 174 are preferably merged such that the resulting ray 174' is essentially an exact continuation of the previous center ray, thus eliminating potential discontinuities present at the region boundaries 170. This is one important advantage of the method of the present invention over known prior approaches. Furthermore, this algorithm can be qualified by characterizing the filtering achieved when adaptively sampling the volume.

An example of a preferred filtering scheme is shown in Figure 20B. Referring to Figure 20B, a Bartlett window (i.e., linear interpolation, triangle filter) is preferably employed. Cascading efficient local Bartlett windows at each region boundary 170 is essentially the equivalent of resampling the rays 174 with a single large Bartlett filter for each baseplane pixel (see Figure 20A). A graphical representation of the preferred filter weights 175 is shown for contribution to the baseplane pixels (e.g., pixels A, B, C).

52

The base sampling rate of the algorithm can be set to a predefined value according to a desired image quality. The base sampling rate is the minimum ray density compared to the underlying volume resolution. Although the ER-Perspective method of the present invention supports virtually any sampling rate, a sampling rate of at least one ray per voxel is preferred. The algorithm has the advantage of keeping the ray density between one to two times the base sampling rate. This guarantees that no voxels are missed in the rear of the volume dataset and places an upper bound on the total amount of work performed at two times (2×) supersampling.

Since the present invention utilizes slice-order processing, the volume dataset is projected onto a baseplane of the volume which is most perpendicular to the view direction. The baseplane image is then warped onto the final image plane in a conventional manner (e.g., in the same manner as in shear-warp or the prior Cube-4 architecture).

The ER-Perspective method of the present invention is ideally suited for implementation on the Cube-5 architecture described above. Specifically, this algorithm preferably only requires nearest neighbor communication between processing elements. While processing a row of voxels on a one-dimensional array of processing elements, the algorithm only requires processing elements to communicate with their immediate left and right neighbors. The Cube-5 rendering pipelines similarly support nearest neighbor communication.

The ER-Perspective algorithm of the present invention preferably employs slice-order processing along one of the three major axes. Consequently, the regions in the ER-perspective algorithm are defined as slabs of slices along a major projection axis. In a preferred embodiment of the ER-perspective method according to the present invention, the volume dataset is projected along slices perpendicular to the z-axis. So as not to limit the methods of the present invention to projections along the z-axis only, it is to be appreciated that the coordinate system may be flipped and the geometry rotated. The algorithm proceeds, as illustrated in Figure 7, by measuring the distance along the z-axis, from the viewpoint 86 to the front of the volume dataset 80,

is determined (e_z). Subsequently, a first region 92 is created to consist of as many z-slices as this distance. Each successive region after the first region 92 is preferably twice as deep as the one before it.

When combined with high quality supersampling, the first region is exactly as large as needed to have one ray per voxel at the end of the region when shooting one ray per pixel of the final image. Thus, supersampling higher than 2× might be needed in the front of the volume to render high quality close up views.

As illustrated in the example of Figure 19, if the viewpoint 172 is three voxel units from the front of the volume (i.e., the z = 3 region boundary), for example, then the first region 176 is preferably three voxel units thick, the next region is six voxel units thick, and so on. In general, the I-th region is preferably $e_z \cdot 2^i$ slices thick, where e_z is the distance from the viewpoint 172 to the front of the volume (see Figure 19). Forcing the regions to be thus defined produces the desired effect that any two perspective rays 174 cast through any of the regions are twice as far apart at the rear boundary (i.e., the z = 24 boundary) as they are at the front boundary (i.e., the z = 3 boundary). This is shown in Figure 19 as the distance between the two rays 174 grows from one unit to two units across the first region 176, then to four units, and finally to eight units at the rear of the last region. Additionally, since the region boundaries 170 are dependent on the global geometry, the efficiency of the ray casting algorithm is maximized by providing a mechanism for keeping the ray density between one and two times the underlying volume resolution in each dimension. It also creates a regular topology so that the filtering of the data can be controlled as perspective rays are cast.

Having regions with boundaries at exponential distances produces a ray density twice as high at the front as at the back of the region. Therefore, a mechanism must preferably be provided to adjust the ray density when crossing a region boundary. Since each ray preferably starts on a voxel coordinate at the rear of a region, at the front of the region every second ray in each dimension will preferably coincide directly with a voxel coordinate. The remaining rays preferably intersect the

region boundary halfway between two voxel positions. To down-sample the ray density with this deterministic ray pattern, a two-dimensional (2D) Bartlett filter (also known as tent or triangle filter) is preferably employed, with an extent of ±1 voxel unit in each dimension. Because the ray density at the front of each region is twice the voxel density, this 3×3 voxel neighborhood is intersected by 5×5 rays. Referring now to Figure 21, since the edges 178 each have a weight of zero, only the 3×3 neighboring rays 180 are used for applying the filter to down-sample the ray density. This effectively merges neighboring rays. A Bartlett filter is preferred over a simple box filter for the added quality it produces in the final image. For the case of front-to-back processing, rays are split instead of merged. Here a bilinear interpolation of the rays is performed to generate the new rays which begin between the other rays. It should be mentioned that a Bartlett filter of size ±1 is the inverse of a bilinear interpolation operation.

Figure 22 shows a 2D example of how sight rays 186 travel through a 7^3 volume 192 when the viewpoint 196 is three voxel units in front of the volume (i.e., from the baseplane 198). Notice that the sampling rate remains between 7 and 14 per slice, and that it increases as the rays 186 travel through the regions from back to front. The number of ray density resampling stages for an N^3 volume is limited by $log_2 N$, since that is the maximum number of regions in an N^3 volume. The last re-sampling step shown on the baseplane 198 is preferably performed when the final image warp takes place.

As illustrated in Figure 22, the rear of the volume dataset 182 does not necessarily always coincide with a region boundary 184. However, since it is preferred that the rays 186 be on exact voxel coordinates 188 at all of the region boundaries 184, the rays 186 preferably originate on the grid coordinates 190 at the rear of the last region enclosing the volume dataset 192 (shaded area). Therefore, the voxel coordinates and the ray sample locations 194 may not be congruent at the rear of the volume 182. This not only provides the mentioned boundary conditions, but aids with temporal anti-aliasing when the viewpoint 196 is moved in smaller than

55

voxel unit distances, because the rays 186 will continue to originate from the same positions relative to the voxels.

Figure 23 depicts a preferred method for performing ER-Perspective back-to-front projection of a volume, in accordance with one form of the present invention, although other embodiments of the ER-Perspective method are contemplated. As described above, first, the distance from the eye or viewpoint to the baseplane is preferably determined (in voxel units). Using this viewpoint position, exponential region boundaries are created. Next, enough regions are preferably established to completely encompass the volume dataset. To perform the volume rendering, the algorithm loops through each region from the back to the front, computing normal ray casting, but in a slice-order fashion, and stores the partially computed rays in a compositing buffer. Between regions (i.e., at the region boundaries), ray density re-sampling of the compositing buffer is preferably preformed, as described previously. The baseplane image is then warped onto the final image plane for display.

With adaptive ray density perspective methods known in the prior art, it is generally difficult to determine the filtering function achieved when rays are merged using irregular patterns. However, since the ER-Perspective method of the present invention preferably uses regular boundaries for the filtering operations and exact ray placement within the boundaries, it is easier to compute the effective filter achieved by the cascading of local Bartlett filters. This is an important advantage of the ER-Perspective algorithm of the present invention. Additionally, the boundaries and filter of the present invention have preferably been chosen to overcome the poor image quality usually associated with conventional successive filtering of discrete data.

Consider, for example, the case of a perspective projection of a volume seven slices deep with the viewpoint two voxel units in front of the volume, as depicted in Figure 24. Using the ER-Perspective method of the present invention, the rays 200 that are cast through a region are one voxel unit apart at the rear of the region.

However, when the rays reach a region boundary 202 they are preferably filtered using local Bartlett filters. The Bartlett filters (simplified to 1-dimension) contain the following weights for a kernel of size $2n+1$, normalized so that the output has the same scalar range as the input:

5

$$0, \frac{1}{n^2}, \frac{2}{n^2}, \dots, \frac{n-1}{n^2}, \frac{n}{n^2}, \frac{n-1}{n^2}, \dots, \frac{2}{n^2}, \frac{1}{n^2}, 0$$

For two-dimensional images at region boundaries, the present invention preferably employs a two-dimensional Bartlett filter by convolving two one-dimensional Bartlett filters in the two principal directions. The ER-Perspective algorithm preferably always resamples the rays to have half of the original density. Using a filter of size ±2

10 rays ($n=2$) creates a filter kernel of 5×5, or just the following five weights for one dimension:

$$0, \frac{1}{4}, \frac{2}{4}, \frac{1}{4}, 0$$

By way of example, as illustrated in Figure 24, consider the contribution of

15 samples a, b, c, d and e to the partially composited ray which changes from region 2 to region 1 at location o,

$$o = \frac{1}{4} b + \frac{2}{4} c + \frac{1}{4} d$$

Likewise, the partial rays at locations p and q are computed as:

$$p = \frac{1}{4} d + \frac{2}{4} e + \frac{1}{4} f$$

$$q = \frac{1}{4} f + \frac{2}{4} g + \frac{1}{4} h$$

The equations for partial rays n and r have been omitted since they have a weight of zero in the final filter for pixel A. Continuing the ER-Perspective algorithm, the resampled partial rays n, o, p, q and r are preferably cast through region 1 where they are again filtered by a local Bartlett filter. The normalized contribution of n, o, p, q and r to pixel A will be:

$$A = \frac{1}{4} o + \frac{2}{4} p + \frac{1}{4} q$$

Substituting in the values for o, p and q results in:

$$A = \frac{1}{16} b + \frac{2}{16} c + \frac{3}{16} d + \frac{4}{16} e + \frac{3}{16} f + \frac{2}{16} g + \frac{1}{16} h$$

It is to be appreciated that this formula contains the same weights (i.e., coefficients) as a Bartlett filter with a kernel size of nine values ($n = 4$). This can be repeated for pixel B with the same filter weights. For front-to-back processing, a similar analysis can be used to demonstrate the performance of the algorithm and the result of successive applications of the bilinear interpolation.

Each sample of a slice preferably contributes the same amount to the final image as any other sample in the same region (assuming all other operations on samples, such as color mapping and compositing, are equal). For example, the value that sample e contributes to pixel A with an effective weight of 1/4 after the cascading of the local Bartlett filters. Likewise, sample l contributes to pixel B with an effective weight of 1/4. Sample f contributes to pixel A with a weight of 3/16 and to pixel B with a weight of 1/16 for a total of 1/4. This can be repeated for samples g and h. The samples to the left of sample e and to the right of sample l partially contribute to pixels left of pixel A and right of pixel B, respectively, such that the sum of their contributions to the final image is also 1/4. In fact, every sample that is in this region

has the same weight. The weight is 1/4 because this region is the second region in the volume. For the first region in the volume, every sample preferably has a weight of ½. This is qualifiable by realizing that there are two rays per final image pixel in this region. There are four rays per final image pixel in the second region, etc.

5 Consequently, the weight which determines the contribution of each sample towards the final image is the ratio $\dfrac{image\ pixels}{samples\ in\ this\ slice}$.

Since the ER-Perspective method of the present invention performs a slice-order processing, the total amount of computation may be analyzed by

10 calculating the amount of work performed on each slice. Assuming that the work done on each sample is the same, the count of the number of samples processed can be used as a comparison of the workloads. For example, in the oversampling method (see Figure 18B), the number of samples on the rear slice of a volume which ends exactly on a region boundary is N^2. On the front slice, the sample count depends on

15 the geometry of the viewpoint. In particular, using similar triangles and defining e_z as the distance from the viewpoint to the front of the volume, the number of samples taken is

$$\left(\frac{N^2 + N \cdot e_z}{e_z} \right)^2$$

This can be generalized for any slice s through the volume dataset to

$$\left(\frac{N^2 + N \cdot e_z}{e_z + s} \right)^2$$

Thus, the total count of samples processed by the oversampling method is

$$\sum_{s=0}^{N} \left(\frac{N^2 + N \cdot e_z}{e_z + s} \right)^2$$

20

Similarly, the undersampling method (see Figure 18A) can be shown to perform the

following amount of work:

$$\sum_{s=0}^{N} \left(\frac{N \cdot e_z}{e_z + s} \right)^2$$

For the ER-Perspective algorithm of the present invention, the analysis is more

complicated. Depending on the viewing geometry, $\log\left(\dfrac{N + e_z}{e_z} \right) - 1$ regions are

5   created. It has been shown previously that each of these regions preferably contains e_z

· 2^i slices. Again, using the geometric principle of similar triangles, the

ER-Perspective algorithm of the present invention processes the following number of

samples:

$$\sum_{reg=0}^{\log\left(\frac{N + e_z}{e_z} \right) - 1} \sum_{s=0}^{e_z * 2^{reg}} \left(\frac{N * (e_z * 2^{reg} - e_z + s)}{e_z * 2^{reg} - e_z} \right)^2$$

This formula has an upper bound of

$$\sum_{s=0}^{N} (2N)^2$$

10   and a lower bound of

$$\sum_{s=0}^{N} N^2$$

Examining the equation for the total count of samples processed by the

oversampling method (given herein above), it can be seen that the oversampling

approach could perform $O(N^4)$ work on the front slice when the viewpoint is very

close to the volume. The oversampling run times grow rapidly as the viewpoint is

15   moved closer to the front of the volume. Examining the undersampling equation

above, it can be seen that as the viewpoint approaches the front of the volume, the

numerator approaches zero. The amount of work performed on the rear slice also

60

approaches zero. The run times of the undersampling method decrease as the viewpoint becomes closer to the volume.

Regardless of the viewpoint geometry, the amount of work performed by the ER-Perspective algorithm of the present invention is bounded by $o(N^2)$ and $\omega(4N^2)$ per slice. Some advantages of this approach are that the upper bound on the run time of the algorithm is linear with the number of voxels and is independent of the view position, and a lower bound on the image quality achieved is also independent of the view position. Thus, a user can set the base sampling rate for the desired image quality and be sure that the sampling rate is sufficient throughout the volume for that desired image quality.

In contrast, a conventional oversampling approach provides a lower bound on the image quality yet the runtime of the algorithm may become much greater than that of the ER-Perspective method of the present invention. A conventional undersampling method provides an upper bound on the runtime for rendering, but the image quality may become much worse than the ER-Perspective approach.

Referring again to Figure 23, a preferred back-to-front ER-Perspective ray-casting algorithm, in accordance with the present invention, is illustrated. The algorithm of Figure 23 is shown as a pseudo-code representation and assumes a Z-major axis projection. The ER-Perspective algorithm of the present invention does not suffer from the traditional pitfalls when performing perspective projections on uniform regular grids. This unique approach runs faster than oversampling methods and produces better quality images than undersampling methods. Employing a Bartlett filter for ray merging provides an image quality improvement over a conventional box filter. The ER-Perspective algorithm is qualified by characterizing the effective filtering on the input data.

In accordance with another form of the present invention, a method is presented for rendering a large volume, wherein the volume dataset exceeds the physical single-pass capacity of the Cube-5 apparatus of the present invention. The

preferred method subdivides the volume dataset into a plurality of cuboid bricks. Traversing the bricks in a predefined order preferably enables initialization of the compositing buffer of the Cube-5 apparatus with a baseplane image of a previous brick before rendering it, whereby ray path and compositing are logically extended throughout the entire volume. Information regarding the boundary between bricks is preferably re-read to insure correct sampling. Using this approach, the maximum volume size is limited only by the available intermediate baseplane storage.

In areas of the dataset where, during perspective projection, multiple voxels contribute to the same image pixel, images of equivalent quality may preferably be rendered using a level-of-detail (LOD) tree, which may be generated, for example, by combining voxels of increasing neighborhood size in a pre-processing step. While perspectively rendering a single large volume utilizing LOD, preferably only a small portion of the volume, substantially close to the viewpoint, must be read in its highest detail. The more distant portions of the volume, with respect to the viewpoint, may then be rendered from lower resolution versions of the data. Thus the frame rate and/or dataset size is preferably increased. Since each region in the perspective algorithm of the present invention (previously described) will now be at a different LOD, there is no longer need to filter the rays between regions, but merely to redistribute them. Preferably, only one region of each LOD tree level is processed; thus, only those regions must be paged into memory.

The level-of-detail (LOD) method of the present invention may also be used for rendering scenes comprised of multiple objects at differing distances from the viewpoint. For such cases, a starting LOD is preferably selected that delivers a baseplane image of about the same size as the screen space image, thereby relating rendering time to image resolution and not to object size (i.e., scale independence).

Although back-to-front rendering is similarly contemplated by and within the scope of the present invention, the unique LOD method will be described herein in a front-to-back rendering context. Rendering front-to-back, it is preferable to start with a slab of the most detailed representation of the volume to be rendered. In a preferred

62

method of the present invention, the thickness of the volume slab is chosen so that projected voxel distances in front and back of the slab differ by a factor of two, similar to perspective projections according to the present invention, as previously described herein. After rendering a slab, the current compositing buffer image is preferably scaled by a factor of 0.5 in the warp unit. This initializes the compositing buffer for the rendering of the next slab of half the resolution. Preferably, only one slab of each LOD actually flows through the rendering pipelines; thus, for large volumes, only those slabs must be paged into the on-board 3D memory.

It is to be appreciated that the apparatus of the present invention can also be employed to speed up off-line computations, such as generation of level-of-detail (LOD) and filtering of datasets. To generate LODs, the trilinear interpolation unit (TriLin) of the present invention preferably sets all its weights to 0.5. Once new samples become available, they are preferably subsampled and compacted into a new volume, which is the next coarser LOD. To filter a dataset, the trilinear interpolation unit again uses only 0.5 weights; this time, however, data is fed back to the beginning of the rendering pipeline without compaction. Each additional pass creates a new filtered volume with a filter kernel having one more voxel extent in every major axis direction.

For higher quality image rendering, the apparatus and methods of the present invention preferably provide the flexibility to utilize a full hardware implementation, multi-pass algorithms, and/or a combination of the two, depending on the desired tradeoffs. The full hardware implementations and multi-pass methods preferably provide more accurate computations in two primary functional areas: filtering and interpolation.

The Cube-4 architecture, a predecessor of the present invention (Cube-5), utilizes a central difference gradient filter with only two sample points to estimate each of the x, y and z gradients at a particular location. A larger 3D filter can deliver a more accurate gradient estimate, such as a Sobel filter (which is a 3^3 filter with weights derived from the inverse of the Manhattan distance from the center point). A

straightforward hardware implementation of a 3^3 filter, however, requires 27

multipliers and 26 adders.

The apparatus of the present invention presents an alternative to this expensive
prior art approach by using symmetric convolution filters. The convolution filters can
be efficiently implemented with only three multipliers and six adders, at a significant
cost savings. Replication of hardware per gradient component can preferably be
avoided by applying a three-pass algorithm instead. As an example, Figure 25
illustrates a symmetric approximation of the x-component of the Sobel gradient filter.
Within each stage, the weights are preferably applied to the nearest neighbors before
summation. With reference to Figure 25, if each stage operates on the output of a
previous stage instead of on the raw data, the weights presented in Figure 25 will
effectively produce the 3^3 symmetric approximation of the Sobel gradient filter (right
side of Figure 25). Changing the x-weights to {1 w 1} will produce an approximation
of a Gaussian filter instead.

The present invention contemplates higher quality rendering modes in which
no additional hardware is needed, but in which the frame rate is lowered. One such
example is to achieve larger neighborhood contributions to the gradient estimation by
utilizing level-of-detail (LOD) information. If the central difference gradient is
computed on data of the next coarser LOD, it is effectively the equivalent of
employing a 6×4×2 filter, with 6 being the extent in the direction of the current
gradient component. Since the apparatus of the present invention (i.e., Cube-5
architecture) is able to hold mip-mapped LOD representations of the data, this filter is
preferably achieved with essentially no increase in hardware, beyond the simple
central difference solution.

Another higher quality multi-pass rendering mode provided by the present
invention, for which no additional hardware is required, is an approximation of tri-
cubic interpolation, which has beneficial applications in the medical field as well as
other fields. This mode enables more accurate resampling and iso-position
calculation. For this, the present invention preferably decomposes a piecewise 4^3-

64

voxel filter into a series of linear interpolations and extrapolations which is symmetric in every dimension, thereby allowing efficient reuse of intermediate results.

In performing higher quality rendering, it is to be appreciated that there are certain tradeoffs between using additional hardware for providing more accurate and flexible gradient estimation within the Cube-5 pipeline, as opposed to employing multiple pass algorithms. Generally, using a multiple pass algorithm requires changes in the Address Generation and Control unit (see Figure 5) of the present invention to momentarily stall the pipeline for computational purposes, while the hardware approach requires additional application specific integrated circuit (ASIC) logic and additional connections to support larger neighborhoods.

With respect to enhanced volume rendering capabilities, a preferred embodiment of the present invention supports clipping by arbitrary planes. The distance from each plane may preferably be incrementally computed using only registers and one adder per plane. In addition to conventional clipping planes which define only the positive direction as visible, the apparatus of the present invention preferably supports extracting an arbitrarily thick slice from the dataset for oblique multi-planar reformatting (MPR) by invalidating all samples lying outside a predetermined offset.

Axis-aligned cutting planes are preferably implemented by restricting the volume traversal to the cuboid of interest. Alternatively, the present invention contemplates restricting this traversal to exclude a simple cuboid from the volume (e.g., visualizing all but one octant of a volume).

In addition to clipping, the present invention further contemplates depth cueing, which modulates the color of objects to simulate, for example, atmospheric attenuation of light through a translucent medium. This phenomenon, as appreciated by those skilled in the art, is termed fog or haze when the medium also contributes some color (e.g., white or gray). To implement this feature in accordance with the present invention, normally clear regions are preferably replaced with a semi-

65

transparent color (e.g., black for depth cueing, white for fog) by modifying the transfer function. Each final pixel is preferably further attenuated to account for the distance from the viewpoint to the surface of the volume, preferably implemented as a part of warping.

The apparatus of the present invention additionally supports rendering of super-sampled images with a preferred default super-sampling rate of two in the x and y directions, although other sampling rates are contemplated. To improve image quality further, the sampling rate along each ray can also be increased. Neither approach requires re-reading voxels from the 3D memory. The apparatus of the present invention preferably changes the volume traversal order so that voxels already residing in the buffers will be read out repeatedly. Each time they are reused, new weights are preferably utilized in the trilinear interpolation units (TriLin) of the present invention to reflect the new resampling position.

In a preferred embodiment for supersampling in the present invention, central difference gradients are computed between neighbors one distance unit apart to ensure sufficient precision. These gradients are preferably computed by taking the difference first and interpolating afterwards or, alternatively, by interpolating first and then taking the difference between neighbors k positions apart (assuming k times oversampling), and preferably not immediate neighbors. A classification stage must consider the new inters ample distances when computing a new α' value. Therefore, during super-sampling, the volume will preferably be traversed in an interleaved pattern within each slice. This essentially ensures that a translucent material (gel) keeps its accumulated opacity (RGBα value) independent of the sampling rate. Thus, for example, for an oversampling factor of k in the z-direction, modified α' values are preferably used, where: $\alpha' = 1 - (1 - \alpha)^{1/k}$.

Anisotropic datasets have different distances between samples along different axes. Thus, the gradient computation and the final two-dimensional (2D) image warp preferably require axis-dependent scaling factors. In addition, the direction in which the sight rays are being cast through the volume dataset preferably require adjustment

66

to account for the implicit volume scaling, which occurs when storing anisotropic data
in an isotropic grid. The α' value is preferably adjusted according to the
direction-dependent distance d which a sight ray travels through a voxel cell. The
corrected α' is $\alpha' = 1 - (1 - \alpha)^d$, with the direction-dependent distance d preferably

5   being in the range $[1, \sqrt{3}]$.

        In addition to the methods for enhancing volume rendering capabilities
described herein above, the present invention further provides several unique methods
for universal three-dimensional (3D) rendering, including mixing polygons and

10   volumes, voxelization of polygons, rendering multiple overlapping volumes,
performing texture mapping and accelerating image-based rendering. These methods
are described in greater detail herein below.

        An important aspect of the present invention is its unique ability to correctly

15   mix geometric objects (i.e., polygons) and volumes in a single image. The apparatus
of the present invention (i.e., Cube-5) preferably leverages conventional geometry
hardware to render opaque and translucent polygons together with the Cube-5 volume
rendering pipeline.

20   In a preferred method according to the present invention, to render a scene
containing volumes and opaque polygons, all opaque polygons are first projected onto
a Z-buffer coincident with a predefined baseplane and having sufficient resolution to
match the volume sample distance. Using the Z-buffer, a determination is preferably
made as to which slices of the volume are in front of the polygons for each pixel of

25   the baseplane image. The compositing buffer is then preferably pre-loaded (i.e.,
initialized) with this projected RGBαZ (i.e., Z-buffer) image, representing the color
and depth image of the polygons. Subsequently, the volume is rendered with z-
comparison enabled in the compositing buffer. The depth values of the opaque
polygons are checked to keep volume samples which are hidden by opaque polygons

30   from contributing to the final image. Ultimately, the opaque polygons occlude the
volume behind, and the volume in front correctly composites over the polygons.

In other words, the compositing buffer is pre-loaded with the z-buffer image $\{C_z, Z_z\}$, in accordance with the preferred method of the present invention, where C_z represents the value of the geometry sample and Z_z represents the depth of the geometry sample from a predetermined viewpoint. During back-to-front compositing, the resulting output pixel in the compositing buffer, C_{out}, will preferably be equal to the geometry sample value, C_z, when the volume sample is behind the geometry (i.e., when the depth of the sample, Z_s, is greater than the geometry depth, Z_z). Similarly, during front-to-back compositing, the samples are preferably composited using the Porter-Duff *over* operator, as appreciated by those skilled in the art. A more detailed discussion of the Porter-Duff α compositing rules are described, for example, in the text Compositing Digital Images, by T. Porter and T. Duff, *Computer Graphics (SIGGRAPH 84)*, vol. 18, no. 3, pp. 253-259, July 1984, which is incorporated herein by reference. Therefore, the resulting output pixel in the compositing buffer, C_{out}, will preferably be equal to the volume sample value, C_s, *over* the geometry sample value, C_z, when the volume sample is in front of the geometry (i.e., when the depth of the volume sample, Z_s, is less than the geometry depth, Z_z).

Translucent polygons pose a more complicated situation, since all fragments (both translucent polygon pixels and volume samples) must be drawn in topologically depth-sorted order. This is required because compositing translucent fragments with the *over* operator is not commutative. Therefore, polygons must be re-depth-sorted whenever the scene or viewing geometry changes. Additionally, the sorting must be topologically correct, including the handling of depth cycles.

Although there are proposed architectures which use an A-buffer to provide some hardware sorting support, implementing an A-buffer in hardware allows only limited depth complexity (i.e., number of overlapping polygons per pixel) in a single pass and is costly. A discussion of a conventional A-buffer algorithm may be found, for example, in the text The A-Buffer, an Antialiased Hidden Surface Method, by L. Carpenter, *Computer Graphics (SIGGRAPH 84)*, vol. 18, no. 3, pages 103-108, July 1984.

In a preferred method, the present invention adapts polygon rendering to slice order ray casting, and synchronizes the overall rendering process on a volume slice-by-slice basis, rather than a polygon-by-polygon or pixel-by-pixel basis. The Cube-5 apparatus preferably utilizes the geometry pipeline and conventional graphics hardware to render geometric objects in thin slabs that are interleaved or dove-tailed between slices of volume samples 212, as illustrated in Figure 26.

With reference now to Figure 26, each slice of the volume is preferably sampled in planes perpendicular to the volume storage axes. The planes are drawn in depth order (e.g., using near and far clipping planes) from farthest from the eye or viewpoint 214 to nearest to the eye. Therefore, to mix translucent polygons with volumetric data, thin slabs of the polygons 210 are preferably rendered and composited in between the slices of volume samples 212. It is to be appreciated that the slabs 210 represent all of the translucent objects which lay between two consecutive slices of the volume sample planes. The boundaries of the slabs are preferably defined such that the union of all rendered slabs 210 neither misses nor duplicates any region (e.g., ( ], ( ], ..., ( ], as shown in Figure 26). The data from the volume slices and the translucent polygonal slabs 210 are dove-tailed together in an alternating fashion. In this manner, the correct depth ordering of all contributing entities is preserved and use of the *over* operator to composite them creates correct colors in the final image pixels.

In accordance with a preferred method of the present invention, the opaque polygons are drawn first with Z-buffering. Before drawing any volume slices, the translucent polygons which lie behind the volume extent are preferably drawn over the opaque polygons using any conventional translucent polygon rendering algorithm (e.g., painters). Likewise, translucent polygons which lie in front of the volume are preferably drawn after the mixing portion of the algorithm. Polygons which lie depth-wise within the volume boundary, but to the top/bottom/side of the volume, are preferably drawn in slice order as if the volume slices were planes that extend to infinity cutting the translucent polygons.

OpenGL may be used to directly render the thin slabs of translucent polygonal objects. The polygons are preferably shaded using the Gouraud shading model included in OpenGL. A naive approach would be to render the complete set of translucent polygons for every slab and set the hither and yon clipping planes to cut the current thin slab of data. However, for an n^3 volume, there could be up to n thin slabs that must be rendered. Since a typical scene contains very few polygons which span all of the thin slabs, the present invention contemplates an alternative approach which would involve clipping the polygons to the slab boundaries and only rendering the portions of the polygons within each slab. This would substantially reduce the processing load on the polygon pipeline. However, it would require the application to clip every polygon against the two planes of each thin slab which contains that polygon.

As illustrated in Figure 27, it is contemplated that the present invention may take advantage of the fact that the two clipping planes 216, 218 are parallel to keep only the portions of the polygons which lie between the planes. While this creates fewer polygons than clipping against each plane separately, it still can increase the triangle count dramatically. The first case occurs when a triangle 220 intersects the thin slab, but no vertices are within the slab boundaries 216, 218. When this occurs, one vertex must be on one side of the slab and the other two vertices on the other side of the slab, thus creating a trapezoid which is decomposed into two triangles. Next, consider when one vertex of a triangle is within the slab. In one situation, a triangle 222 intersects the slab such that the remaining two vertices lay on the same side of the current slab, creating only one triangle. In a second situation, a triangle 224 intersects the slab such that the remaining two vertices lay on opposite sides of the current slab. This is a worst case situation, since it produces a pentagon, or three triangles. The final case occurs when a triangle 226 intersects the slab such that two vertices lie within the same slab and, once again, a trapezoid is created resulting in two triangles.

In a preferred embodiment of the present invention, a bucket sorting method is applied to the translucent polygons. Whenever the viewing geometry changes, the placement of volume sample planes change their relative positions to the geometry.

Therefore, the present invention preferably creates a bucket for each thin slab between two volume sample planes. All of the translucent polygons in a scene are preferably traversed and each of the polygons is placed in a bucket for each of the slabs it intersects. For example, as shown in Figure 28, triangle T1 is placed in all six buckets since it spans all six slabs S1-S6. Triangle T2 is placed in buckets corresponding to slabs S2 and S3, and likewise for the remaining triangles. For the example shown in Figure 28, bucketing the four triangles T1 - T4 would result in twelve triangles being sent to the graphics pipeline. As a comparison, if the triangles were being clipped to the slab boundaries, twenty triangles would be sent to the graphics pipeline.

An alternative to bucketing is to create an active triangle list similar to the active edge list utilized in scan converting polygons. The triangles may be placed in the active list at the first slice they intersect and removed from the list when they no longer intersect any slices. A data structure is preferably pre-computed which indicates which slice each triangle first encountered. This preprocessing is essentially the same as for bucketing, with the exception that bucketing does not have to check for triangle removal for each slice.

One advantage of the method of the present invention is that for applications which choose to trade off image quality in order to maintain a predetermined frame rate, the number of polygons drawn decreases as the number of slices drawn for the volume decreases. This occurs because the interslice size increases as the number of volume slices decreases. The rendering rate achieved is substantially proportional to the number of polygons drawn and the number of volume samples drawn (which is proportional to the number of volume slices drawn). The image quality degradation resulting from this tradeoff affects only the volume data, similar to taking fewer samples in any volume rendering algorithm.

When mixing translucent geometries and volumes, there exist at least three options for handling two or more translucent polygons being drawn to the same pixel within one thin slab. In the first option, the polygons could be drawn in regular processing order with the *over* operator. While this method may produce the incorrect

color, the amount of color error is limited because the polygons are still sorted by bucketing them into thin slabs.

5   Another method for handling two or more translucent polygons is to draw thin slabs of translucent polygons between two volume sample slices as on-the-fly voxelization. In conventional voxelization methods, when a surface is 3D scan converted into a 3D volume grid, the resolution of the grid is commonly chosen such that the size of a single voxel represents the smallest area that can be discerned by the human eye when it is rendered. In the X and Y dimensions, the polygons are drawn to

10   screen resolution. In the Z dimension, it is assumed that the volume is being rendered with enough slices such that each volume sample also represents the smallest area that can be discerned by the human eye. Therefore, each pixel bounded by two volume slices in the Z dimension also represents this small area.

15   In view of the foregoing, a method, performed in accordance with one embodiment of the present invention, may be viewed as computing on-the-fly voxelization by utilizing 3D graphics hardware. Voxelization methods combine polygons into a single voxel by using one of two preferred methods. The first method is to take the *max* of each color channel. The second method is to take the weighted-

20   *max* as

$$C_v = \frac{(C_{p1}D_{p1} + C_{p2}D_{p2})}{(D_{p1} + D_{p2})}$$

where C_{p1} is the color of a first polygon (polygon 1), D_{p1} is the density of polygon 1, and C_v is the color assigned to the voxel. Many OpenGL implementations, for example, allow *max* blending with *glBlendEquationext(gl_max_ext)*. Assuming that the density is equal to the alpha value (e.g., linear ramp transfer function for volume

25   rendering), then the colors may preferably be weighted by their alpha values before blending by using a *glBlendFunc (gl_src_alpha, gl_one)*. However, OpenGL is not able to compute the complete previous equation since it cannot divide by the sum of the alpha values after accumulating them.

The third method of drawing two or more translucent polygons to the same pixel within one thin slab may also be considered the most accurate approach. By utilizing one of the previously described methods of the present invention to perform depth sorting, such as BSP tree, proper ordering of all translucent polygons within each slab is maintained. Depth cycles are preferably handled by the BSP algorithm by splitting polygons which span a plane used in the partitioning, and eventually one of the polygons in the cycle is used as the partitioning plane.

As previously discussed, an important feature of the present Cube-5 invention is the unique ability to couple at least one geometry pipeline or engine to the Cube-5 system. In accordance with the present invention, two preferred methods of connecting one or more geometry pipelines to the claimed Cube-5 system on PC-class machines is provided, as described herein below. Both methods allow the unique mixing of opaque and/or translucent polygons with volumetric data.

It is to be appreciated that the opaque polygons are preferably rendered such that, after projection through the volume dataset, warping creates the correct footprint on the final image. Furthermore, the Z-depth values are preferably aligned along the processing axis, so that a volume slice index may be used for the Z-depth check.

In accordance with one embodiment of the present invention, a preferred method begins by determining a major viewing axis for the current viewing direction. As illustrated in Figure 29, a transformation is preferably applied to the geometry 228 so that the major viewing axis 230 is along, for example, the Z-axis. Next, the view or eye point 232 is moved to be along this direction, preferably by rotating the vector between the look-at point 234 and the eye point 232 by a predefined angle α around the X-axis and an angle β around the Y-axis. Preferably, α and β are always in a range between -45 and $+45$ degrees, otherwise a different baseplane would be chosen. A Z-slice shear transformation along X and Y (also known as a "X and Y according to Z" shear) is preferably subsequently applied to the viewing matrix as follows:

73

$$\begin{bmatrix} 1 & 0 & \tan\alpha & 0 \\ 0 & 1 & \tan\beta & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

With this geometry, when the opaque polygons are drawn, the polygon footprints are "prewarped" so that the warping operation at the end of Cube-5 rendering creates correct polygons in the final image. Additionally, the Z-depths computed are preferably proportional to the distances along the processing axis. It is possible (e.g., if all opaque geometry fits within the volume extents) to set the hither and yon clipping planes to the edges of the volume and, if the precision of the depth buffer is the same, the depths computed are exactly the volume slice indexes for depth checking. Otherwise, a simple scaling must be applied when the computed depths are utilized by the volume rendering system. Light positions should be considered when using this method, however, as the shearing may not move the lights to the correct location.

The thin slices of translucent polygons preferably align geometrically with their 3D positions in space. Preferably, the eye point is first aligned as previously described. Next, in order to keep the objects from projecting all the way to the final image plane, the geometry is preferably translated such that the center of the current thin slab is at the Z=0 plane prior to shearing. Clipping planes allow only the current thin slab to be rendered and the projection plane is set to be within the two volume slices which border that region with, for example, *glOrtho* (*glFrustum* for Perspective).

Important to comprehending the present invention is to understand the organization of frame buffer design and compositing buffer design. As illustrated in Figure 30, the Cube-5 volume rendering pipeline 236 of the present invention preferably utilizes a tightly coupled on-chip SRAM buffer 238, termed a compositing buffer, to hold the partially composited rays as a volume is processed in slice order.

74

This architecture exploits the regular processing sequence inherent in slice order rendering. Specifically, each slice of the volume 240 is preferably processed in the same order as the previous, left-most voxel to right-most voxel of each row, and bottom-most row to top-most row of each slice (possibly with some skewing). In this way, the SRAM composting buffer 238 becomes a simple FIFO queue having a length equal to the size of a slice. The SRAM queue is preferably 32 bits wide to hold 8-bit fixed point RGBα values (called coxels). Each pipeline 236 preferably reads a coxel from the front of the queue and writes a coxel to the rear of the queue for each clock cycle.

In contrast, with reference now to Figure 31, conventional PC-class geometry pipelines 242 utilize an external DRAM frame buffer 244, which stores the RGBα color values and Z-depth values for each pixel. This frame buffer 244 must support random access, since polygon rendering does not enjoy the regular access ordering inherent in slice-order volume rendering. Normal polygon rendering produces triangles on a screen of average between 10 and 50 pixels. Therefore, the DRAM memory is organized to maximize access to areas of the screen of this size.

As shown in Figure 31, when the 3D texture mapping method of the present invention is implemented on geometry pipelines 242, volume slices 246 perpendicular to the screen are texture mapped through the volume. The per-vertex geometry calculations for the volume slices 246 are easily achievable with any level graphics hardware. However, the requirement to support random access to both the texture memory 248 and frame buffer 244 limits the performance of this approach to the fill rate achievable with a DRAM frame buffer.

Very high end surface graphics systems typically utilize massive parallelism in the fragment processing section 250 of the polygon pipeline. This, coupled with a highly distributed frame buffer, allow increased fill rate performance.

In Figure 32 there is shown one embodiment for connecting a geometry pipeline 242 to the Cube-5 volume rendering system 252, according to the present

invention. As illustrated in Figure 32, the SRAM composting buffer is preferably removed from inside the Cube-5 pipeline 252 and replaced with an external DRAM frame buffer 254. Rather than organizing the DRAM frame buffer 254 as in conventional polygon engines, the memory in the frame buffer of the present invention is preferably organized so that it is specifically optimized for volume rendering. The frame buffer 254 is also preferably accessible from a 3D graphics pipeline 242 to allow mixing of polygonal data 256 with volumes.

With continued reference to Figure 32, the dual use frame buffer 254 preferably connects the two pipelines 242, 252. In a preferred method, to render a scene with both opaque and translucent polygons and also volume data, the geometry pipeline 242 first renders all opaque polygons with Z-depth. The volume slices, stored in volume memory 258, and thin slabs of translucent polygons are then rendered in an alternating (e.g., dovetailing) fashion - volume slices by the Cube-5 pipeline 252 and translucent polygons by the graphics pipeline 242 (opaque polygons may also be rendered with the same dovetailing algorithm, but with increased demand on the graphics pipeline).

Z-depth checking is preferably utilized to insure correct hidden object removal and blending is set in both pipelines to correctly composite the samples and fragments. The geometry engine 242 preferably performs the final baseplane warp required by the Cube-5 system of the present invention.

The design of the DRAM buffer 254 is critical to achieve, for example, the 503 Million samples per second required for 30Hz rendering of 256^3 volume datasets. Therefore, it is helpful to first create a DRAM buffer for the Cube-5 rendering pipeline itself, before discussing connecting the rendering pipeline to a graphics pipeline. The volume rendering system of the present invention is preferably comprised of multiple Cube-5 pipelines. In each rendering pipeline, at every clock cycle, a coxel (composting buffer element consisting of RGBα) is read from the SRAM composite buffer FIFO and blended with an appropriate composting equation. The new coxel is then placed at the rear of the FIFO. In a preferred embodiment, the

structure of a coxel is changed to contain 32 bits of color, 8 for each RGBα and 32 bits of Z-depth information, 24 + 8-bit stencil. This configuration is required to handle Z-depth checking in the composting stage. Assuming that opaque polygon rendering is completed before any volume rendering begins, the 32 bits of Z-depth/stencil information is read, but not re-written. Therefore, for every clock cycle, each Cube-5 pipeline needs to read 8 bytes of coxel data and write back 4 bytes.

Preferably, the rendering pipeline of the present invention utilizes memory chips with a word size of 16 bits. Using this configuration, four words must be read by each pipeline every cycle and two words must be written. To do this would require six 16-bit memory interfaces per pipeline. An emerging technology in synchronous DRAM (SDRAM) chips, which the present invention may avail itself, is known as double data rate (DDR), which reads/writes data at both the rising and falling edges of the clock. Using DDR SDRAMs, the present invention can utilize two 16-bit memory interfaces for reading 64 bits of data per clock and one 16-bit memory interface for writing 32 bits per clock, for a total of three 16-bit memory interfaces per pipeline.

With reference now to Figure 33, since a read and write must be performed every clock cycle in order to keep the pipeline full, the present invention preferably reads from one set of frame buffer chips (e.g., set A) 260 and writes to another set (e.g., set B) 262. The Cube-5 system contemplates reading from set A 260 and writing to set B 262 for a complete slice of the volume, and then swapping for the next slice. With this approach, however, each frame buffer chip set would have to be large enough to hold the complete frame buffer. Furthermore, the polygon engine would have to be instructed as to which set is the current set. Therefore, in a preferred embodiment, the present invention alternates reading and writing between set A 260 and set B 262 within a frame and buffers the processed coxels from the read set until it becomes the write set. Since every memory access must be a burst, each burst actually lasts four clock cycles and reads/writes four coxels (i.e., eight words) with 16-bit DDR DRAM chips. The Cube-5 system preferably cycles through all 4 banks to keep the memory bandwidth saturated before writing the new RBGα values back.

77

For this reason, there is preferably a 16-coxel FIFO queue 264 (four coxels for each of four banks) that the newly composited RBGα portions of the coxels are stored in.

5   There are many different possible configurations for the number of pipelines etc. in the Cube-5 volume rendering system of the present invention. An example for a case of four parallel pipelines creating 12 total memory interfaces will now be discussed with reference to Figure 33. As shown in Figure 33, each pipeline contains one read interface 266 to the Z-depth/stencil portion 268 of the frame buffer and two read/write interfaces 270 and 272 to set A 260 and set B 262, respectively, of the

10   RGBα portion of the frame buffer. To render a 256^3 volume at 30Hz, each of the four pipelines process 125 million voxels per second. Therefore, a 133 MHZ clock is utilized for the chip and the SDRAM. The mapping of the frame buffer pixels onto the memory chips is critical to the performance. It must match exactly the processing order of the Cube-5 pipelines and the parallel access by four pipelines substantially

15   simultaneously. It is assumed that the skewed memory access of the Cube-5 architecture is "un-skewed" so that the volume samples are in order from left to right across each scanline in groups of four, since it is easier to follow in the explanations. The design can be extended to skewed memory, although the geometry pipeline and screen refresh system must be aware of the additional skewing.

20

Figure 34 shows a preferred layout of the RGBα portion of the coxels in the frame buffer. For a given scanline 274, there is a group of pixels which reside in set A 276 followed by a group of pixels which reside in set B 278, repeated across the entire scanline 274. The length of each set is 64 pixels due to the fact that each set

25   must contain pixels which are read from four different banks inside each chip, and each bank consists of four RGBα values from four parallel chips/pipelines. Thus the pixel data in the frame buffer is interleaved across eight chips; In fine detail, it is really interleaved across only four chips. This provides an interface which reads

30   4 pipelines × (1 RGBα chip + 1 depth chip) × 16 bits
       × 133MHz × 2 data rate = 34 Gbits = 4.15Gbytes

of data per second. This surpasses the required

$$256^3 \times 30\text{Hz} \times 8 \text{ bytes} = 3.75\text{Gbytes per second}$$

where eight bytes are organized as four bytes RGBα + four bytes Z-depth/stencil. Additionally, the frame buffer sub-system is capable of writing

$$4 \text{ pipelines} \times 1 \text{ RGBα chip} \times 16 \text{ bits} \times 133\text{MHz}$$
$$\times 2 \text{ data rate} = 17\text{Gbits} = 2.1\text{Gbytes}$$

once again handling the

$$256^3 \times 30\text{Hz} \times 4 \text{ bytes} = 1.8\text{Gbytes per second}$$

required for real time 30Hz rendering of 256^3 volumes.

This extra bandwidth is not siting idle. The screen must be refreshed from the data in the frame buffer. If we assume a 1280 × 1024 screen resolution with 60Hz refresh rate and that all four RGBα bytes are read from the frame buffer (since our burst mode access retrieves them anyway), then

$$1280 \times 1024 \times 60\text{Hz} \times 4 \text{ bytes} = 300\text{Mbytes}$$

are read from the frame buffer per second. Only the RGBα portion of the frame buffer is required for refresh. Therefore, the refresh data is read from eight chips. It is sufficient to perform ten data burst reads/writes (depending on set A or set B) to each chip followed by one read of data for refresh. This distribution of memory accesses provides the refresh hardware with a consistent (although bursty) stream of data. The Cube-5 pipelines also contain a small percentage of excess cycles, and thus will not lose the ability to achieve 30Hz 256^3 rendering when the memory sub-system is temporarily stalled for refresh.

An alternative approach to connecting a graphics pipeline to the Cube-5 volume rendering pipeline, in accordance with a preferred embodiment of the present invention, will now be described. This preferred connection approach keeps both the graphics pipeline and the volume rendering pipeline working at all times and merges the data in the SRAM compositing buffer inside the Cube-5 chip. At any given time, the volume rendering pipeline is composting the current volume slice with the previous thin slab of polygon data over the compositing buffer and the graphics pipeline is rendering the next thin slab of translucent polygons.

The method described herein still utilizes the unique approach of dovetailing volume slices and thin slabs of translucent polygonal data, as previously described herein above. In a first step, all opaque polygons are projected onto a Z-buffer coincident with the baseplane (e.g., the volume face most parallel to the screen). Next, the projected RGBαZ image is loaded into the composting buffer of the volume rendering pipeline. Subsequently, the volume is rendered with a Z-comparison enabled in the composting stage. The thin slabs of translucent polygons are preferably rendered by the geometry pipeline, and their corresponding RGBα data is sent to the volume pipeline of the present invention to be blended into the SRAM composting buffer within the volume pipeline.

Preferably, the composting stage of the volume rendering accelerator is modified to composite two layers (one volume and one translucent polygon) per step, thus not delaying the volume rendering process. This requires the addition of some extra logic. The straightforward formula for performing a double composition of a volume sample v over a translucent pixel fragment p over the old coxel c would require four additions and four multiplies in five stages:

$$C_c = C_v\alpha_v + [C_p\alpha_p + C_c(1 - \alpha_p)](1 - \alpha_v)$$

However, employing simple math allows the double composition to be calculated with four additions and two multiples in six stages with the following formula (some of the calculations are re-used):

$$C_c = (C_c + (C_p - C_c)\alpha_p) + [C_v - (C_c + (C_p - C_c)\alpha_p)]\alpha_v$$

80

As appreciated by one skilled in the art, the hardware designer would choose the option more desirable for a given implementation (i.e., less logic and more stages, or fewer stages and more logic).

5   Consider the amount of data transferred for a 256^3 volume. There are preferably 255 slabs plus one buffer in front of the volume and one buffer behind the volume. Each of these 257 slabs contains 256KB (256^2 pixels of RGBα) of data. This equates to 64MB being read from the frame buffer and transferred between the two sub-systems each frame. To achieve a 30Hz frame rate would require a

10   bandwidth of 1.9GB per second. While this much data could be transferred with sufficiently wide channels, it must also be read from the frame buffer. It would be virtually impossible to read this much data without changing the organization of the current DRAM frame buffers. Additionally, the frame buffer must be cleared 257 times per frame.

15

       To solve this bandwidth challenge, the present invention preferably uses run-length encoding (RLE) of the blank pixels. With this method, each scanline is encoded separately and a "run-of-zeros" is encoded as four zeros (RGBα) followed by the length of the run. Since typically only a small percentage of the polygons in a

20   scene are translucent, the translucent polygon slabs will be relatively sparse. Run-length-encoding just the blank pixels in these thin slabs results in over 99% reduction in the required bandwidth. Preferably, the method of the present invention utilizes RLE on 2D images of sparse translucent polygons to save on bandwidth.

25   Using this preferred method requires adding hardware to the Cube-5 system of the present invention. Specifically, additional hardware may be included in the volume rendering pipeline that can decode the RLE input stream and create RGBα fragments. However, since these fragments are utilized by the volume pipeline in a regular order, it is preferable to decode the input stream using a double buffer to

30   synchronize the two pipelines. Every clock cycle, a value is output from the decoding hardware. If the volume rendering machine has multiple pipelines (as most current

81

designs do) the decoding hardware is preferably replicated for each pipeline so that
they can keep up with pixel demand.

5   Likewise, RLE hardware at the originating end connected to the geometry
pipeline may encode the data in real-time before sending it to the volume pipeline.
However, 1.9GB per second access to the frame buffer would still be required to read -
all the thin slabs of translucent polygons and clear the frame buffer 257 times per
frame. Therefore, a separate frame buffer is preferably employed which stores the
data directly in RLE format. Since the thin slabs of translucent data are very sparse,
10   more time is spent clearing and reading than is spent rasterizing. An RLE buffer,
while generally not optimal for rasterization, is well suited for both clearing and
reading the data. For example, to clear an RLE frame buffer requires merely storing a
single run of zeros (in five bytes) for each scanline, instead of writing an entire 256^2
frame buffer.

15

         To minimize the impact on the current geometry pipelines, the RLE frame
buffer is preferably implemented using the emerging technology of embedded DRAM
and connecting it in parallel to the normal frame buffer. This differs from
conventional encoding algorithms which typically assume that the data was given in
20   physical order. Triangle rasterization, however, does not guarantee any ordering of   -
the fragments. Therefore, the apparatus of the present invention must be able to
randomly insert an RGBα value into an RLE scanline of data.

         Figure 35 illustrates a diagram of an RLE insert, formed in accordance with
25   the present invention. For each fragment, the encoded scanline is copied from one
buffer to another, inserting the new RGBα value. Every clock cycle, a single flit (i.e.,
either an RGBα pixel or run-of-zeros) is processed. The entire scanline is preferably
processed flit by flit. With reference to Figure 35, an input buffer ("in Buffer") 280
holds the current encoded scanline and an output buffer ("out Buffer") 282 holds the
30   newly encoded scanline with the new RGBα fragment inserted. The choice of what to
insert at each cycle is preferably performed by a 5-byte multiplexor 284. The
apparatus of the present invention preferably includes pointers, namely "inPtr" 286

and "outPtr" 288, which point to the current flit of both the in buffer 280 and out

buffer 282, respectively. The logic on the right side of Figure 35 calculates how much

has been processed ("Total") 290 and two of the control points ("ctrl_1" and "ctrl_3").

The other mux control point ("ctrl_2") is calculated by 'OR'-ing together all of the

5   RGBα values (the flag for run-of-zeros). "xPos" is defined as the x position of the

fragment. Preferably, a lookup table is implemented of where the current buffer is

located in memory for each y value. Thus, the buffer can be moved while inserting

new pixels and the table is simply updated. This preferred method is illustrated in the

RLE_AddFragment pseudo-code routine of Figure 36. With continued reference to

10   Figure 36, the RLE_AddPixelToScanline function demonstrates the processing that

occurs in the hardware embodiment of the present invention shown in Figure 35.

By utilizing an embedded DRAM the present invention takes advantage of the

extremely high bandwidth available when processing occurs on the memory chip.

15   The processing is simple enough to be implemented in the DRAM manufacturing

process. For example, for a 1280 × 1024 frame buffer, the maximum amount of

memory required is 50Mbits. This fits onto eDRAM dies with room for over 3

million gates for the encoding hardware.

20   Figure 37 is a preferred block diagram illustrating how a polygon pipeline 242

and volume pipeline 252 are connected through the RLE frame buffer 292, which is

preferably double-buffered to allow rendering during transmission of data. The

auxiliary frame buffer is preferably connected at the same place as the existing one by

simply duplicating the fragments, thus not affecting the remainder of the geometry

25   pipeline 242. The volume pipeline 252 also preferably utilizes double buffering to

allow receiving of data while blending the previous slab. It is to be appreciated that,

using the system of the present invention, volume rendering does not conflict with

polygon rendering. Since the volume pipeline 252 always accesses its memory in a

repeatable ordered fashion, it achieves the sample fill rate into the frame buffer at a

30   sufficient rate to achieve 30Hz volume rendering. The system of the present invention

utilizes the graphics pipeline 242 to render the opaque polygons before rendering the

volume, stored in volume memory 258. This can normally be accomplished

concurrently with the rendering of the volume for the previous frame. Even if the polygon engine must render translucent polygons mixed in with the volume, there is usually enough time to render the opaque polygons before the volume finishes due to the small number of translucent polygons in normal scenes.

In accordance with a preferred embodiment of the present invention, a method is provided to incrementally voxelize triangles into a volumetric dataset with pre-filtering, thereby generating an accurate multivalued voxelization. Multivalued voxelization allows direct volume rendering with intermixed geometry, accurate multiresolution representations, and efficient antialiasing. Prior voxelization methods either computed only a binary voxelization or inefficiently computed a multivalued voxelization. The method, in accordance with the present invention, preferably develops incremental equations to quickly determine which filter function to compute for each voxel value. This preferred method, which is described in greater detail herein below, requires eight additions per voxel of the triangle bounding box.

To avoid image aliasing the present invention preferably employs pre-filtering, in which scalar-valued voxels are used to represent the percentage of spatial occupancy of a voxel, an extension of the two-dimensional line anti-aliasing method conventionally known (Filtering Edges for Grayscale Displays, by S. Gupta and R. F. Sproull, *Computer Graphics (SIGGRAPH 81)*, vol. 15, no. 3, pp. 1-5, Aug. 1981). It has also been shown that the optimal volume sampling filter for central difference gradient estimation is a one-dimensional oriented box filter perpendicular to the surface. The method of the present invention preferably utilizes this filter which is a simple linear function of the distance from the triangle.

Conventional graphics hardware only rasterizes points, lines, and triangles, with higher order primitives expressed as combinations of these basic primitives. Therefore, it is preferable to voxelize only triangles because all other primitives can be expressed in terms of triangles. Polygon meshes, spline surfaces, spheres, cylinders, and others can be subdivided into triangles for voxelization. Points and lines are special cases of triangles and can similarly be voxelized by the present

84

algorithm. To voxelize solid objects, the boundary of the object is preferably

voxelized as a set of triangles. The interior of the object is then filled using a

volumetric filing procedure.

5   As appreciated by those skilled in the art, *edge functions* are linear expressions

that maintain a distance from an edge by efficient incremental arithmetic. The

methods of the present invention extend this concept into three dimensions and apply

antialiasing during the scan conversion of volumetric triangles.

10   In essence, the general idea of the triangle voxelization method of the present

invention is to voxelize a triangle by scanning a bounding box of the triangle in raster

order. For each voxel in the bounding box, a filter equation is preferably evaluated

and the result is stored in memory. The value of the equation is a linear function of

the distance from the triangle. The result is preferably stored using a fuzzy algebraic

15   union operator, namely, the *max* operator.

           With reference now to Figure 38, there is shown a density profile of an

oriented box filter along a line 294 from the center of a solid primitive 296 outward,

perpendicular to the surface 298. The width of the filter is defined as W. The

20   inclusion of a voxel in the fuzzy set varies between zero and one, inclusive,

determined by the value of the oriented box filter. The surface 298 of the primitive

296 is assumed to lie on the 0.5 density isosurface. Therefore, when voxelizing a

solid primitive 296, as in Figure 38, the density profile varies from one inside the

primitive to zero outside the primitive, and varies smoothly at the edge. For a surface

25   primitive, such as the triangle 300 shown in Figure 39, the density is preferably one

on the surface and drops off linearly to zero at distance W from the surface. Although

the present invention similarly contemplates the voxelization of solids, the

voxelization of surfaces will be described herein.

30   With continued reference to Figure 39, it has been determined that the

optimum value for filter width W is $2\sqrt{3}$ voxel units (see e.g., Object Voxelization by

Filtering, by M. Šrámek and A. Kaufman, *1998 Volume Visualization Symposium*, pp.

111-118, IEEE, Oct. 1998). For shading, the normal is preferably estimated by computing the central difference gradient at the 0.5 isosurface. Because the overall width of the central difference filter is at most $2\sqrt{3}$ units, a correct gradient is found on the 0.5 density isosurface. The thickness of the triangle 300 may be defined as T.

5   Normally, T can be zero, unless thick surfaces are desired. By thresholding at 0.5 density, a 6-tunnel-free set of voxels is generated when $W \geq 1$. This property is useful for volumetric filling (e.g., in order to generate solid objects).

10   All voxels with non-zero values for a triangle are preferably within a bounding box which is $S = W + T/2$ voxel units larger in all directions than a tight bounding box. Therefore, the first step of the present method preferably determines a tight bound for the triangle 300, then inflates it in all directions by S voxel units and rounds outward to the nearest voxels.

15   As illustrated in Figure 40, the area surrounding a triangle defined by vertices C_1, C_2 and C_3 may be divided into seven regions (e.g., R1 through R7) which must be treated separately. In a preferred method of the present invention, each candidate voxel is tested for inclusion within the seven regions, then filtered with a different equation for each region. In the interior region R1 of the triangle, the value of the

20   oriented box filter is simply proportional to the distance from the plane of the triangle. In regions along the edges of the triangle, R2, R3, R4, the value of the filter is preferably proportional to the distance from the edge of the triangle. In regions at the corners of the triangle, R5, R6, R7, the value of the filter is preferably proportional to the distance from the corner of the triangle.

25

With continued reference to Figure 40, the regions R1 - R7 are preferably distinguished by their distance from seven planes. The first plane a is preferably coplanar with the triangle and its normal vector **a** points outward from the page. The next three planes b, c, and d preferably have normal vectors **b**, **c**, and **d** respectively

30   and pass through the corner vertices C_1, C_2, and C_3 of the triangle, respectively. The final three planes e, f, and g are preferably perpendicular to the triangle and parallel to the edges; their respective normal vectors, **e**, **f**, and **g**, lie in the plane of the triangle

86

and point inward so that a positive distance from all three planes defines region R1.
All of the plane coefficients are normalized so that the length of the normal is one,
except for normal vectors **b**, **c**, and **d** which are normalized so that their length is
equal to the inverse of their respective edge lengths. In that manner, the computed
5   distance from the plane varies from zero to one along the valid length of the edge.

For any planar surface, the distance of any point from the surface can be
computed using the plane equation coefficients:

$$Dist = \frac{Ax + By + Cz + D}{\sqrt{A^2 + B^2 + C^2}}$$

10

which simplifies to

$$Dist = Ax + By + Cz + D$$

when the coefficients are pre-normalized. This computation can be made incremental
so that when stepping along any vector, the distance only changes by a constant. For
15   example, if the distance from a plane is *Dist* at position [*x, y, z*], then stepping one unit
distance in the *X* direction changes the distance to

$$Dist' = A(x + 1) + By + Cz + D$$
$$= Ax + By + Cz + D + A$$
$$= Dist + A$$

In general, stepping along any vector **r** = [r_x, r_y, r_z], the distance from the plane
changes by

$$Dist' = Dist + r \odot [A, B, C]$$

20   where $\odot$ indicates the dot product. While scanning the bounding box of the triangle,
the distance from the plane of the triangle can be computed incrementally with just a

single addition per voxel. This method, performed in accordance with the present invention, for computing the distance from a plane is illustrated by the preferred pseudo-code routine shown in Figure 41.

5   The Y-step is more complicated than the X-step because it not only steps one unit in the Y direction, but it also steps back multiple units in the X direction. Consider, as an analogy, the operation of a typewriter which glides back to the left margin of the paper and advances the line with one push of the return key. Similarly, the Z-step combines stepping back in both the X and Y directions and stepping

10   forward one unit in the Z direction. This simple pre-processing step ensures efficient stepping throughout the entire volume. If numerical approximation issues arise, then it is possible to store the distance value at the start of each inner loop and restore it at the end, thereby minimizing numerical creep due to roundoff error in the inner loops.

15   For multivalued voxelization, seven plane distances are required. Therefore, seven additions are required per voxel to compute the plane distances. Other computations per voxel may include incrementing the loop index, comparisons to determine the appropriate region and, if necessary, computations to determine the density.

20

Referring again to Figure 40, in region R1 the density value of a voxel is preferably computed with the box filter oriented perpendicular to plane a. Given a distance $DistA$ from plane a, the density value V is computed using:

$$V = 1 - \frac{|DistA| - T/2}{W}$$

25   In region R2, the density is preferably computed using the distance from planes a and b:

$$V = 1 - \frac{\sqrt{DistA^2 + DistB^2} - T/2}{W}$$

Similarly, region R3 uses planes *a* and *c*, and region R4 uses planes *a* and *d*. Region R5 uses the Pythagorean distance from the corner point C_1:

$$V = 1 - \frac{\sqrt{(C_1^x - x)^2 + (C_1^y - y)^2 + (C_1^z - z)^2} - T/2}{W}$$

Similarly, regions R6 and R7 use corner points C_2 and C_3, respectively.

5

At the shared edge of adjacent triangles, it is preferable to avoid discontinuities or cracks. Fortunately, the oriented box filter guarantees accurate filtering of the edges for any polyhedra, provided the union of the voxelized surfaces is correctly computed. The union operator can be defined over multivalued density

10 values $V(x)$ with $V_{A \cup B} \equiv \max(V_A(x), V_B(x))$. Other Boolean operators are available. However, the *max* operator preserves the correct oriented box filter value at shared edges, and is therefore preferred.

The implication of using *max* in the method of the present invention is that the

15 current voxel value must be read from memory, then possibly modified and written back into memory. Therefore, a maximum of two memory cycles are required per voxel.

The efficiency of the algorithm of the present invention may be further

20 increased by limiting the amount of unnecessary computation because the bounding box often contains a higher percentage of voxels unaffected by the triangle than affected by it. The bounding box can be made tighter by recursively subdividing the triangle when edge lengths exceed a predetermined constant.

25 To visualize intermixed polygons and volumes, the polygons are preferably voxelized into the target volume and rendered in a single pass. If the polygons move with respect to the volume, then voxelization should occur into a copy of the original volume so as not to corrupt the data. The multivalued voxelized polygon voxels may

be tagged to distinguish them from volume data. In this manner, polygons can be colored and shaded separately from other data.

The preferred triangle voxelization algorithm described above is efficiently implemented in the distributed pipelines of the Cube-5 volume rendering system of the present invention. This algorithm adds just a small amount of hardware to the existing pipelines and performs accurate multivalued voxelization at interactive rates. One important advantage of the claimed Cube-5 volume rendering algorithm is that the volume data is accessed coherently in a deterministic order. This feature allows orderly scanning of a bounding box for this algorithm.

In Figure 42, a preferred embodiment of the overall voxelization pipeline is shown, in accordance with the present invention. If on-the-fly voxelization is important, the system of the present invention may preferably include separate pipelines for volume rendering and voxelization. If voxelization can occur in a separate pass, then these volume rendering and voxelization pipelines may be combined, with the voxelization pipeline re-using most of the hardware from the volume rendering pipeline. The setup for each triangle preferably occurs on the host system, in a similar manner as setup is performed on the host system for 2D rasterization.

With reference to Figure 42, in the first hardware stage 302 of the pipeline, the distances from the seven planes are preferably computed. Seven simple distance units are allocated with four registers for each of the seven planes. Preferably, one register holds the current distance from the plane and the other three registers hold the increments for the X-, Y-, and Z-steps. Figure 43 shows a distance computation unit 310 for one of the seven planes, formed in accordance with a preferred embodiment of the present invention. This distance computation unit 310 may be included as part of the distance calculation stage 302 of the pipeline (see Figure 42). The other six units can be essentially identical in design, but hold different values. During each clock cycle of voxelization, the pipeline preferably steps in either the X, Y, or Z direction (i.e., performs an X-Step 312, Y-Step 314, or Z-Step 316), thereby updating the

current distance according to the direction of movement. The hardware for looping through the volume is already present in the volume rendering pipeline and is therefore re-used here to scan the bounding box of the triangle.

After the seven plane distances are calculated, the resulting values preferably flow down the pipeline. As shown in Figure 42, the next pipeline stage 304 then preferably determines in which region the current voxel resides. In a preferred embodiment of the region selection stage 304, only seven comparators are needed to determine the outcome of the truth table, due to the mutual exclusion of some cases. For instance, in Figure 40, from the negative (lower) side of plane b, it is not necessary to test the distances from plane f or g, depending on the value of the distance from plane e.

With continued reference to Figure 42, after the region has been determined, the next pipeline stage 306 computes the filter function. The filter calculation stage 306 of the pipeline is preferably only activated if the current voxel is within S voxel units of the triangle. Otherwise, the current voxel is essentially unaffected by the triangle and different regions require different calculations, ranging from a simple linear expression to a complex Pythagorean distance evaluation. Since hardware ideally must handle all cases equally well, it is preferred that such hardware be able to perform a square root approximation by means of a limited resolution look up table (LUT). However, the range of inputs and outputs is small, and therefore the size of the required LUT will be small. Furthermore, the Cube-5 hardware of the present invention has several LUTs available for volume rendering which can be re-used for voxelization. Instead of providing three separate units to compute the expression $V = 1 - (\sqrt{Dist} - T/2)/W$, it is more efficient to roll all the calculations into one LUT. In this case, the input is $Dist^2$, defined over [0,12], and the output is the density value V in the range [0,1].

Due to the mutual exclusion of the seven regions, it is sufficient to provide hardware for only the most complex filter calculation. With reference to Figure 40, the most complex calculation is the corner distance computation of regions R5, R6,

91

and R7 which, in a preferred embodiment, requires five adders and three multipliers, in addition to the square root LUT previously mentioned. The line distance computations in regions R2, R3, and R4 are simpler, requiring only one adder, two multipliers and the square root LUT. Region R1 requires a single multiply to obtain the distance squared, which is the required input to the LUT.

Referring again to Figure 42, the final stage 308 of the pipeline preferably computes the *max* operation using the current voxel value and the computed density estimate. In a preferred embodiment of the present invention, the *max* operator is simply a comparator attached to a multiplexor such that the greater of the two values is written back to memory. Since most voxels in the bounding box are not close enough to the triangle to be affected by it, memory bandwidth will be saved by only reading the necessary voxels. Further bandwidth savings may be achieved by only writing back to memory those voxels that change the current voxel value. Since there is some latency between requesting and receiving word from memory, the voxel is preferably fetched as soon as possible in the pipeline and the results queued until the memory is received. The final stage 308 is write-back to memory, which can be buffered without worry of dependencies.

The present invention thus far has been described outside the context of skewing, which complicates the traversal. However, the present invention contemplates building skewing into the Y- and Z-step distance update values. Skewing also adds more complexities to the Cube-5 hardware of the present invention. Specifically, when a left-most voxel moves one unit in the Y direction, placing it outside of the bounding box, the pipeline actually takes $p - 1$ steps in the X direction to keep the voxel within the bounding box. Similarly, when the left-most voxel moves one step in the Z direction, it also moves one step in the negative X direction, which is handled in the same way as before. Therefore, the apparatus of the present invention is preferably adapted to perform skewing by adding fourteen (14) more registers and corresponding logic to determine when the pipeline is currently processing the left-most voxel.

92

Pre-filtering, which may be performed in combination with the voxelization methods of the present invention, can be used to optimally generate a series of volumes of different resolutions. This technique is useful for rendering images of different sizes; the size of the volume is preferably chosen to correspond to the size of the final image. In this manner, aliasing is avoided at all image resolutions and no unnecessary work is performed rendering parts of a scene not visible at the image scale.

Pre-filtering can additionally be used to model motion blur. For example, as an object sweeps past a camera, it sweeps out a complex volume during the time the shutter is open, causing motion blur. To accurately render motion blur, conventional rendering techniques render multiple images and blend them into a single image. However, this approach is very slow. With pre-filtering, the present invention performs the sweeping operation once, during voxelization, so that motion blur can be rendered in the same time as regular volume rendering. This method works well, particularly for certain cases where the motion is constant (e.g., the same direction and/or rotation). For example, consider a helicopter blade which spins at a constant speed during flight. For example, to voxelize the blade spinning at the rate of 5Hz for an animation frame rate of 30Hz, the blade sweeps out an arc of $\frac{5}{30}(2\pi)$ each frame. Thus, at the outer edge of the blade, the density value is much lower and the blade appears more transparent than in the center, where it sweeps out a smaller volume and appears more solid. The volume rendering transfer function may be set so that the lower density values appear less opaque and higher density values appear more opaque.

When multiple volumetric objects overlap, the projected image of the volumes becomes quite complex. Consider, for example, a scene where smoke rises up through a cloud. Clearly, the two volumetric objects cannot be rendered separately with the images combined in the final frame. Therefore, in a preferred method, performed in accordance with one form of the present invention, multiple objects are combined into one object for a final rendering pass to create the resulting image.

When two or more objects occupy the same space, the colors from each object are preferably modulated together at each sample location along a projected sight ray. Therefore, it is preferred that each object be classified and shaded prior to being combined, followed by color modulation. If, alternatively, voxel data were combined first, a new transfer function would be required for each possible combination. This latter approach is therefore not preferred.

In accordance with one form of the present invention, a preferred method for mixing multiple overlapping volumes resamples all but the first object in the z-dimension of the first object so that slices of each object become interlaced. This includes a classification, a shading and a transformation which aligns all objects. Object transformations include translation and scaling, preferably performed by the apparatus of the present invention using nearest neighbor connections, and rotation, which is preferably performed using the rotation methods of the present invention previously described herein above.

For scenes containing objects which will not change position or orientation with respect to each other, the present invention contemplates optimizations such as high-level scene graph compilation that can preferably be employed. For instance, static objects are preferably combined once and stored for subsequent rendering, while non-static objects are re-combined each time they are moved with respect to the other objects.

Texture mapping is a widely used technique to simulate high-quality image effects, such as surface details, and even lighting and shadows. In general terms, texture mapping involves mapping a two-dimensional (2D) image onto a three-dimensional (3D) surface. Texture mapping occurs while geometric objects are rasterized onto the screen. The (x, y) pixel coordinates are preferably mapped into (u, v) texture coordinates and an RGBα value is returned as the color value to use for that pixel on the screen.

There are basically two processes involved in texture mapping: a mapping

from (x, y) coordinates to (u, v) coordinates, and a look-up into the image of what

RGBα value corresponds to a given (u, v) coordinate. The mapping from (x, y) to $(u,$

$v)$ coordinates preferably involves simple matrix multiplication, as appreciated by

5   those skilled in the art. However, the look-up into the image of the (u, v) coordinate

to return an RGBα value is complex. The very large scale integration (VLSI)

hardware requirements for the texture lookup commonly consume large portions of

today's graphics boards, at a significant cost. This is primarily due to the fact that $(u,$

$v)$ coordinates rarely map directly to a discrete image coordinate, called a *texel*.

10   Therefore, the neighboring RGBα values are preferably linearly interpolated to

produce the RGBα value at the exact (u, v) coordinate.

Two-dimensional (2D) interpolations are generally sufficient if the pixel does

not cover more than one texel. However, if the mapping produces pixel coverages

15   greater than one texel, artifacts are introduced into the image using the 2D

interpolation method. To avoid costly texel combining operations, a technique termed

Mip-Mapping may be utilized by conventional graphics pipelines. Mip-Mapping

basically consists of storing multiple levels-of-detail (LOD) of an image. Then, when

an (x, y) pixel is mapped to a (u, v) texel, the appropriate Mip-Map level texels are

20   chosen so that the pixel is smaller than the texels. A more accurate method is to look-

up the four neighborhood texels from both the higher level and lower level of detail

texel images and then perform a trilinear interpolation on all eight texels to compute

the appropriate RGBα value for the pixel.

25   Texture mapping hardware from conventional graphics pipelines has been

used to accelerate volume rendering and has been the subject of such texts as

RealityEngine Graphics, by K. Akeley, *Computer Graphics (SIGGRAPH 93)*, 27:109-

116, Aug. 1993, and Accelerated Volume Rendering and Tomographic

Reconstruction Using Texture Mapping Hardware, by B. Cabral, N. Cam and J.

30   Foran, *Symposium on Volume Visualization*, pp. 91-98, Oct. 1994. This conventional

approach, however, neither achieves the cost-performance nor supports the various

functionalities (e.g., shading) of the present invention. Furthermore, using known

95

prior art methods, texture mapping is unscalable without data replication, often

employs two-dimensional (2D) rather than three-dimensional (3D) interpolation,

downloads datasets slowly, and/or does not support real-time four-dimensional (4D)

input.

5

In accordance with a preferred form of the present invention, described

previously herein above, the Cube-5 apparatus is combined with a conventional

geometry engine via the geometry input/output bus 46, 48 (see Figure 4). Preferably,

the rendering pipeline(s) of the present invention are utilized to perform the texture

10   look-up function, while the geometry engine is used for mapping (x, y) pixel

coordinates to (u, v) texture coordinates. In simple terms, once combined with the

Cube-5 apparatus, the responsibility of the geometry engine is essentially to rasterize

triangles, while the apparatus of the present invention preferably provides the high

performance interpolation engine for texture mapping. To perform texture look-ups

15   on the apparatus of the present invention, texel data is preferably loaded into 3D

memory included within the Cube-5 unit(s). Figures 6A and 6B illustrate an example

of how 32 bits of texel data for a 2×2 neighborhood are preferably arranged in a 2^3

subcube of 16-bit voxels.

20   Another important advantage of the present invention is the ability to enhance

image-based rendering. In general, image-based rendering methods render complex

scenes from arbitrary viewpoints based on a finite set of images of that scene. Two

similar image-based rendering methods, known by those skilled in the art, which use

four-dimensional (4D) interpolation without requiring the depth information of the

25   source images are light field rendering and Lumigraph. The high-performance

interpolation engine of the present invention may be used to accelerate these two

techniques.

Figure 44 shows that in light field rendering, the scene is modeled by uv 322

30   and st 320 planes. Every uv grid point preferably defines a viewpoint and has an

associated st image. For every pixel of the projection plane 324, a sight ray 326 is

preferably cast into the uv plane 322. The four st images corresponding to the uv grid

points surrounding the intersection of the sight ray with the *uv* plane contribute to that

ray. The contributions are preferably calculated by casting a sight ray into each *st*

image through its *uv* grid point. These rays hit between *st* image pixels and, therefore,

a bi-linear interpolation must be performed for each *st* image. One final bi-linear

5   interpolation between the four rays yields the final projection plane pixel color.

Obtaining every pixel of the projection plane 324, therefore, requires four bi-linear

interpolations in *st* planes 320 and one bilinear interpolation in the *uv* plane 322,

resulting in a total of five bi-linear interpolations. These five bi-linear interpolations

are substantially equivalent to one 4D interpolation, or 15 linear interpolations.

10

Performing lookups for each projection plane ray usually causes random

access into the *st* images. Therefore, in accordance with a preferred method of the

present invention, *st* images are accessed in object order, which is more appropriately

adapted for use with the apparatus of the present invention since the Cube-5 apparatus

15   allows reading of each *st* image pixel only once. With continued reference to Figure

44, for each quadrilateral 328 in the *uv* plane (e.g., *abcd*), its projections on the four *st*

planes (e.g., corresponding to *abcd*) preferably determine which four tile regions 330

contribute to the final image. All *st* tile regions 330 are then preferably assembled

into four images and are perspectively projected onto the projection plane 324. The

20   final image is subsequently formed by bilinear interpolation among the four projected

images. Interpolation weights are preferably determined by the intersection between

the original ray and the *uv* plane 322.

Although illustrative embodiments of the present invention have been

25   described herein with reference to the accompanying drawings, it is to be understood

that the invention is not limited to those precise embodiments, and that various other

changes and modifications may be effected therein by one skilled in the art without

departing from the scope or spirit of the present invention.

WHAT IS CLAIMED IS:

1.   An apparatus for real-time volume processing and universal three-dimensional (3D) rendering, said apparatus being responsive to viewing and processing parameters which define a viewpoint, said apparatus generating a 3D volume projection image or new modified volume from said viewpoint, said image including a plurality of pixels, said apparatus comprising:

        a plurality of three-dimensional (3D) memory units which store a plurality of discrete voxels, each of said voxels having a location and voxel data associated therewith, said voxels together forming a volume dataset, said viewing and processing parameters defining at least one base plane of said volume dataset and first and last processing slices of said volume dataset;

        at least a first pixel bus for providing global horizontal communication;

        a plurality of rendering pipelines, each of said plurality of rendering pipelines being vertically coupled to both a corresponding one of said plurality of 3D memory units and said at least first pixel bus, each of said plurality of rendering pipeline having horizontal communication with neighbors, said rendering pipelines receiving said voxel data from said corresponding memory units and generating a two-dimensional (2D) base plane image aligned with a face of said volume dataset;

        at least one geometry bus for providing global horizontal communication between said plurality of rendering pipelines and a geometry engine, said at least one geometry bus enabling the rendering of geometric and volumetric objects together in a single image; and

        a control unit which initially designates said first processing slice as a current slice of sample points, and which controls sweeping through subsequent slices of said volume dataset as current slices until said last processing slice is reached.

2.   The apparatus of Claim 1, further comprising a feedback bus operatively connected to said plurality of 3D memory units and said plurality of rendering pipelines, said feedback bus providing communication between an output of said rendering pipelines and at least one of:

        an input of said plurality of rendering pipelines; and

said plurality of corresponding 3D memory units.

3.   The apparatus of Claim 2, further comprising an imagery input bus operatively coupled to said plurality of 3D memory units and providing global horizontal communication between said 3D memory units.

4.   The apparatus of Claim 3, further comprising a plurality of two-dimensional (2D) memory units for storing said two-dimensional (2D) base plane image, each of said 2D memory units being operatively coupled to both a corresponding one of said plurality of rendering pipelines and said at least first pixel bus.

5.   The apparatus of Claim 4, further comprising at least one warp unit operatively coupled to said at least first pixel bus, said at least one warp unit assembling and transforming said base plane image onto a predefined image plane.

6.   The apparatus of Claim 1, wherein each of said plurality of rendering pipelines comprises:

at least a first slice unit having an input which is operatively coupled to said corresponding one of said plurality of 3D memory units and having an output, said slice unit including said current slice of sample points, said slice unit receiving voxel values from said 3D memory units;

a compositing unit having an input which is operatively coupled to said output of said at least first slice unit and having an output which is operatively coupled to said at least first pixel bus; and

a compositing buffer having an input which is operatively coupled to said compositing unit and having an output, said compositing buffer storing a plurality of pixels, said pixels having at least one of a color and an opacity associated therewith.

7.   The apparatus of Claim 6, further comprising:

at least a second and a third slice unit, said at least first, second and third slice units buffering said voxels read from said plurality of 3D memory units, wherein subsequent re-reading of said voxels is performed from said slice units.

8.   The apparatus of Claim 7, wherein said control unit organizes a traversal of said processing slices in a block oriented, interleaved fashion, whereby sample points adjacent in time differ by one unit along a major projection axis.

9.   The apparatus of Claim 6, wherein each of said plurality of rendering pipelines further comprises:

        a bilinear interpolation unit operatively coupled to said output of said compositing buffer and said compositing unit;

5   a trilinear interpolation unit having an input which is operatively coupled to said output of said at least first slice unit and having an output;

        a gradient computation unit having a first input operatively coupled to said output of said at least first slice unit and a second input operatively coupled to said output of said trilinear interpolation unit, and having an output; and

10   a shading unit having a first input operatively coupled to said output of said gradient computation unit and a second input operatively coupled to said output of said trilinear interpolation unit, and having an output operatively coupled to said input of said compositing unit.

10.   The apparatus of Claim 9, wherein each of said plurality of rendering pipelines further comprises a texture map bypass bus which operatively couples said trilinear interpolation unit and said gradient computation unit to said bilinear interpolation unit.

11.   The apparatus of Claim 9, wherein each of said plurality of rendering pipelines further comprises at least one feedback connection between said at least one pixel bus and said corresponding one of said plurality of 3D memory units, said feedback connection being configured for feedback from said at least one pixel bus to said 3D

5   memory unit, and subsequently to any intermediate stage in said plurality of rendering pipelines.

12.   The apparatus of Claim 9, wherein said gradient computation unit includes:

100

a multiplier; and

an adder;

10 wherein along each major projection axis a weighted average of three consecutive

sample points is computed and written back through a feedback connection

operatively coupled to said rendering pipeline and said corresponding 3D memory

unit.

13.   The apparatus of Claim 1, wherein said control unit divides a view frustum of

said volume dataset into a plurality of regions separated by region boundaries, said

regions including a first region situated closest to said viewpoint, each successive

region beginning with said first region having a depth along a major projection axis

5 which is twice that of an adjacent region preceding it.

14.   The apparatus of Claim 13, wherein:

a plurality of continuous sight rays are cast through said volume dataset from

said viewpoint, said plurality of sight rays diverging with respect to each other as they

pass through said volume dataset; and

5   said control unit enables one of:

(I)   merging of at least two of said sight rays; and

(ii)   splitting of at least one of said sight rays.

15.   A method for performing perspective volumetric ray casting of a three-

dimensional (3D) volume dataset, said volume dataset including a plurality of voxel

grid point locations and having a baseplane defining a face of said volume dataset

which is most perpendicular to a view direction, said method utilizing discrete voxels

5 stored in a distributed fashion in a plurality of three-dimensional (3D) memory units,

each of said voxels including a location and voxel data associated therewith, said

method comprising the steps of:

(a)   calculating a distance along a major projection axis from said

viewpoint to said baseplane of said volume dataset;

10   (b)   dividing said volume dataset into a plurality of consecutive regions

separated by region boundaries, said region boundaries being situated at exponentially

101

increasing distances from said viewpoint along said major projection axis, said

plurality of regions at least encompassing said volume dataset, wherein each

successive region after said first region has a depth along said major projection axis

15   that is twice the depth of a preceding adjacent region;

(c)   casting a plurality of continuous sight rays extending from said

viewpoint through said volume dataset, said sight rays passing through a current

sample region;

(d)   performing one of:

20   (I)   merging of at least two of said sight rays; and

(ii)   splitting of at least one of said sight rays; and

(e)   repeating steps (c) and (d) by sequentially sampling subsequent regions

of said volume dataset, beginning with said first processing slice, until said last

processing slice is reached, each of said subsequent regions in turn becoming said

25   current sample region.

16.   The perspective ray casting method of Claim 15, wherein:

in step (c), said plurality of sight rays are cast through said volume dataset

such that each of said sight rays coincides with said voxel grid point locations in said

volume dataset at said region boundaries.

17.   The perspective ray casting method of Claim 15, wherein:

in step (d), said merging is performed by a filter having a predefined filter

weight and said splitting is performed by linear interpolation.

18.   The perspective ray casting method of Claim 17, wherein said filter is a

Bartlett filter.

19.   The perspective ray casting method of Claim 15, wherein:

in step (d), a plurality of said sight rays are merged such that a new sight ray is

generated, said new sight ray being an exact continuation of a center sight ray of said

merged sight rays.

102

20.   A method for generating a level-of-detail (LOD) representation of a source
volume dataset, said volume dataset comprising a plurality of discrete voxels stored in
a distributed fashion in a plurality of three-dimensional (3D) memory units, each of
said voxels having a location and voxel data associated therewith, said method
5   comprising the steps of:

    (a)   resampling said volume dataset to generate a plurality of new samples; -

    (b)   discarding every other sample of said plurality of new samples to form
a second volume dataset which is more compact in size than said source volume
dataset;

10   (c)   writing slices of said second volume dataset into said 3D memory
units, thereby replacing said source volume dataset with said second volume dataset;
and

    (d)   repeating steps (a) through (c) until said second volume dataset is a
predetermined size.

21.   The method of Claim 20, wherein the resampling of step (a) is performed
using linear interpolation.

22.   The method of Claim 20, wherein the resampling of step (a) is performed by
trilinear interpolation with weights set to 0.5 in an x, y and z direction.

23.   A method for rendering a level-of-detail (LOD) representation of a volume
dataset, said volume dataset comprising a plurality of discrete voxels stored in a
distributed fashion in a plurality of three-dimensional (3D) memory units, each of said
voxels having a location lying on a gridpoint in said volume dataset and having voxel
5   data associated therewith, said method comprising the steps of:

    (a)   selecting viewing and processing parameters which define:

        a viewpoint; and

        a starting level-of-detail (LOD), said starting LOD producing a
baseplane image substantially equal to a screen space image size, said baseplane
10   image defining a face of said volume dataset which is most perpendicular to a view
direction; and

103

(b)   dividing said volume dataset into a plurality of slabs, each of said plurality of slabs having a thickness along said view direction such that projected voxel distances in front and back of said slabs differ by a factor of two, said plurality of slabs including a starting slab of a most detailed representation of said volume dataset;

(c)   rendering said starting slab into a compositing buffer, said compositing buffer storing a compositing buffer image representing said rendered slab;

(d)   compacting said compositing buffer image by a factor of 0.5;

(e)   writing said scaled compositing buffer image into said compositing buffer;

(f)   repeating steps (c), (d) and (e), by sequentially reading each subsequent slab of said plurality of slabs of the volume dataset at half resolution until all slabs of said volume dataset have been processed.

24.   The method of Claim 23, further comprising the step of providing a warp unit, wherein step (d) is performed by said warp unit.

25.   A forward warping method for performing a geometric transformation between a source image comprising a plurality of source pixels and a target image comprising a plurality of target pixels, said source image being generated from a viewpoint, said forward image warping method comprising the steps of:

selecting viewing and processing parameters which define said viewpoint and a plurality of scanlines;

determining a parallel-preserving scanline direction, wherein parallel scanlines in said source image remain parallel in said target image;

mapping said source image to said target image along said parallel-preserving scanline direction by incrementally sweeping through subsequent scanlines of said source image until all of said scanlines are mapped.

26.   The forward image warping method of Claim 25, wherein said parallel-preserving scanline direction is derived from a perspective transformation matrix.

27.   The forward image warping method of Claim 25, further including the steps
of:

sweeping said scanlines through said source image, wherein samples along
each scanline are incrementally calculated;

reading said source pixels in a fixed pattern according to a predetermined read
template; and

writing sample values of said source pixels according to a predetermined write
template.

28.   The forward image warping method of Claim 27, further including the step of
computing a footprint by calculating distances to neighboring samples in said source
image, wherein samples along said parallel-preserving scanline direction are equi-
distant and wherein a distance of each source pixel to a nearest neighbor target pixel is
computed incrementally.

29.   A method for rendering a scene including at least one volume and a plurality
of geometries, and producing an image, said image being generated from a viewpoint,
said method comprising the steps of:

(a)   providing a plurality of geometry primitives, each of said geometry
primitives having an opacity associated therewith, and a volume dataset including a
plurality of voxels, each of said voxels having a location and voxel data associated
therewith;

(b)   selecting viewing and processing parameters which define:
said viewpoint;

at least one baseplane of said volume dataset, said baseplane defining a
face of said volume dataset which is most perpendicular to a view direction; and

first and last processing slices of said volume dataset;

(c)   separating, according to said opacity, opaque geometry primitives from
translucent geometry primitives;

(d)   rendering said opaque geometry primitives into a Z-buffer, said Z-
buffer having sufficient resolution to match a volume sample distance and storing a Z-
buffer image representing a distance along said view direction from said viewpoint

and an associated sample value for each of said geometry primitives, said Z-buffer

image being on a plane coincident with said first processing slice of said volume

20   dataset;

(e)   initializing a compositing buffer with said Z-buffer image of said

opaque geometry primitives;

(f)   casting a plurality of sight rays through said first processing slice of

said volume dataset;

25   (g)   determining whether a depth of a current sample of said volume

dataset, taken along the view direction, is greater than the depth of said Z-buffer

image;

(h)   compositing said opaque geometry primitive over said volume sample

if the depth of said volume sample is greater than the depth of said opaque geometry

30   primitive; and

(i)   repeating steps (f) through (h) by sweeping through subsequent slices

of said volume dataset until said last processing slice is reached.

30.   The method of Claim 29, further comprising the steps of:

rendering said translucent geometry primitives into a plurality of thin slabs,

each slab of said plurality of slabs having bounds selected such that a union of all

slabs neither misses nor duplicates any region of said volume dataset;

5   compositing said thin slabs of said translucent geometry primitives between

two adjacent slices of said volume dataset until said last processing slice is reached.

31.   The method of Claim 30, wherein data from said volume slices and said

translucent polygon slabs are dove-tailed together in an alternating fashion.

32.   The method of Claim 30, wherein the step of rendering the translucent

geometry primitives comprises the steps of:

creating a plurality of buckets, each bucket corresponding to a thin slab

between two adjacent slices of said volume dataset; and

5   traversing all translucent geometry primitives in the scene and placing each

translucent geometry primitive in a bucket for each slab said translucent geometry

primitive intersects.

33.   The method of Claim 30, wherein the step of rendering the translucent

geometry primitives comprises the step of maintaining an active display list of pre-

rendered translucent geometry primitives, whereby each of said translucent geometry

primitives is placed in said active list at the first slice of said volume dataset the

5   translucent geometry primitive intersects and removed from said active list when the

translucent geometry primitive no longer intersects any slices of the volume dataset.

34.   A method for performing an arbitrary three-dimensional rotation of a volume

dataset with respect to a predetermined viewpoint, said volume dataset comprising a

plurality of discrete voxels stored in a distributed fashion in a plurality of three-

dimensional memory units, each of said voxels having a location lying on a gridpoint

5   in said volume dataset and having voxel data associated therewith, said method

comprising the steps of:

received a target three-dimensional rotation matrix defining a final orientation

of said volume dataset in relation to said viewpoint, said rotation matrix being

expressed as a concatenation of axis rotations along an x, y and z axis; and

10   decomposing said rotation matrix into a sequence of at least two passes of

consecutive shear transformations of said volume dataset.

35.   The three-dimensional rotation method of Claim 34, wherein the

decomposition step comprises four passes of consecutive two-dimensional (2D) shear

transformations in the sequence set forth:

performing a first 2D beam shear transformation of the volume dataset along a

5   first major direction axis;

performing a second 2D beam shear transformation of the volume dataset

along a second major direction axis;

performing a third 2D beam shear transformation of the volume dataset along

a third major direction axis; and

10   performing a fourth 2D beam shear transformation of the volume dataset along

said first major direction axis.

36.   The three-dimensional rotation method of Claim 34, wherein the

decomposition step comprises four passes of consecutive two-dimensional (2D) shear

transformations in the sequence set forth:

performing a first 2D slice shear transformation of the volume dataset along a

5   first major axis direction;

performing a second 2D slice shear transformation of the volume dataset along

a second major axis direction;

performing a third 2D slice shear transformation of the volume dataset along a

third major axis direction; and

10   performing a fourth 2D slice shear transformation of the volume dataset along

said first major axis direction.

37.   The three-dimensional rotation method of Claim 36, wherein the

decomposition step comprises two passes of consecutive three-dimensional (3D)

beam shear transformations, a first pass being a concatenation of said first and second

2D slice shear transformations and a second pass being a concatenation of said third

5   and fourth 2D slice shear transformations.

38.   The three-dimensional rotation method of Claim 34, wherein the

decomposition step comprises three passes of consecutive two-dimensional (2D) shear

transformations in the sequence set forth:

performing a first 2D beam-slice shear transformation of the volume dataset,

5   wherein said shear transformation is a product of a beam shear transformation and a

slice shear transformation, each along a different major axis direction;

performing a second 2D beam-slice shear product transformation of the

volume dataset, wherein said shear transformation is a product of a beam shear

transformation and a slice shear transformation, each along a different major axis; and

10   performing a 2D beam shear transformation of the volume dataset along the

same major axis direction as the beam shear in said first 2D beam-slice shear

transformation.

39.   The three-dimensional rotation method of Claim 34, further comprising the

step of providing a plurality of barrel shifters, wherein said shear transformations of

said decomposition step are implemented by said barrel shifters.

40.   The three-dimensional rotation method of Claim 34, further comprising the

step of providing a plurality of logarithmic shifters, wherein said shear

transformations of said decomposition step are implemented by said logarithmic

shifters.

41.   The three-dimensional rotation method of Claim 34, further comprising the

steps of:

       (a)   setting up a one-to-one correspondence between a source voxel in said

source volume dataset and a target voxel in a target volume dataset, said target

5   volume dataset corresponding to the final orientation of the source volume dataset;

       (b)   calculating, for each of said source voxels, said corresponding target

voxel using said one-to-one correspondence;

       (c)   storing each of said target voxels in said source voxel location in the

source volume dataset by resampling;

10   (d)   transforming said target voxels to corresponding destinations in said

target volume dataset.

42.   The three-dimensional rotation method of Claim 41, wherein:

       said resampling in step (c) is performed by interpolation on local voxels and a

sampling position of each of said target voxels is computed using a backward

transformation of said rotation matrix; and

5   said transformation in step (d) is performed using a nearest neighbor

placement scheme.

43.   A method for rendering a scene including a plurality of volumes and

producing an image, said image being generated from a viewpoint, said method

comprising the steps of:

        (a)   providing a first volume dataset and a first transformation matrix

5   corresponding to said first volume dataset;

        (b)   providing a second volume dataset and a second transformation matrix

corresponding to said second volume dataset;

        (c)   selecting viewing and processing parameters which define:

                said viewpoint; and

10   first and last processing slices of said first and second volume datasets;

        (d)   transforming said first and second volume datasets according to said

first and second transformation matrices, respectively;

        (e)   casting a plurality of sight rays along a view direction from said

viewpoint through said first processing slice of said first volume dataset;

15   (f)   casting a plurality of sight rays along said view direction from said

viewpoint through said first processing slice of said second volume dataset;

        (g)   writing said first slice of said first volume into a compositing buffer;

        (h)   compositing said first slice of said second volume onto said

compositing buffer; and

20   ,   (i)   repeating steps (e) through (h) by sequentially sweeping through

subsequent slices of said first and second volume datasets along said view direction

until said last processing slices of said first and second volume datasets are reached.

44.   The method of Claim 43, wherein step (d) further includes the step of

calculating a difference between said first and second transformation matrices,

wherein only one of said first and second volume datasets is transformed according to

said difference.

45.   The method of Claim 44, wherein step (d) further includes the step of

resampling said volume dataset, if a shear transformation is performed on said volume

dataset, to form a corrected volume dataset, said corrected volume dataset

substantially eliminating errors introduced by said shear transformation.

46.   The method of Claim 45, wherein said resampling step is performed during said ray casting of at least one of steps (e) and (f).

47.   A method for rendering a scene including a plurality of volumes and producing an image, said image being generated from a viewpoint, said method comprising the steps of:

(a)   providing a first volume dataset and a corresponding first transformation matrix, and a second volume dataset and a corresponding second transformation matrix, each of said first and second volume datasets comprising a plurality of discrete voxels stored in a distributed fashion in a plurality of three-dimensional memory units, each of said voxels having a location lying on a gridpoint in said volume dataset and having voxel data associated therewith;

(b)   selecting viewing and processing parameters which define:
        said viewpoint; and
        first and last processing slices of said first and second volume datasets;

(c)   casting a plurality of sight rays along a view direction through said first processing slice of said first and second volume datasets;

(d)   creating a new volume dataset by combining corresponding values of the voxels from said first and second volume datasets at each sample point along said sight rays into new values, each of said new values including a modulated color value; and

(e)   repeating steps (c) and (d) by sweeping through subsequent slices of said volume datasets until said last processing slice of said first and second volume datasets is reached, thereby rendering said new volume dataset as a single volume dataset relative to said viewpoint.

48.   The method of Claim 47, wherein step (d) further includes the step of calculating a maximum value of each of said corresponding voxel values from said first and second volume datasets, and wherein the combining of voxels is performed by compositing using said maximum value as said corresponding new value in said new volume dataset.

49.   The method of Claim 47, wherein step (d) further includes the step of calculating a difference value between each of said corresponding voxel values from said first and second volume datasets, and wherein the combining of voxels is performed by compositing using said difference value as said corresponding new

5   value in said new volume dataset.

50.   The method of Claim 47, wherein step (d) further includes the step of calculating a minimum value of each of said corresponding voxel values from said first and second volume datasets, and wherein the combining of voxels is performed by compositing using said minimum value as said corresponding new value in said

5   new volume dataset.

51.   The method of Claim 47, wherein step (d) further includes the steps of:
       classifying and shading said first and second volume datasets by assigning each of the voxels of said first and second volume datasets a predetermined RGBα value; and

5   combining corresponding voxels of said first and second volume datasets by modulating said RGBα values of corresponding voxels to form an RGBα volume dataset.

52.   The method of Claim 51, wherein step (d) further includes the step of creating an index field corresponding to each of said voxels in said new volume dataset, said index field storing data indicating which volume dataset each voxel in said new volume dataset originated from.

53.   A method for performing high quality rendering of an image comprising the step of computing at least one of an x-gradient component, a y-gradient component and a z-gradient component, wherein:
       said x-gradient component comprises the sub-steps of:

5   (i)   applying a weighted average filter in an x-direction with weights -1, 0, 1;

112

      (ii)   applying the weighted average filter in a y-direction with weights 1, w, 1; and

      (iii)   applying the weighted average filter in a z-direction with weights 1, w, 1;

said y-gradient component comprises the steps of:

      (i)   applying a weighted average filter in an x-direction with weights -1, 0, 1;

      (ii)   applying the weighted average filter in a y-direction with weights 1, w, 1; and

      (iii)   applying the weighted average filter in a z-direction with weights 1, w, 1; and

said z-gradient component comprises the steps of:

      (i)   applying a weighted average filter in an x-direction with weights -1, 0, 1;

      (ii)   applying the weighted average filter in a y-direction with weights 1, w, 1; and

      (iii)   applying the weighted average filter in a z-direction with weights 1, w, 1.

54.   A method for connecting a plurality of Cube-5 rendering pipelines to a geometry pipeline, said method comprising the steps of:

providing a plurality of external memory units, said memory units having a bandwidth sufficient to feed a screen refresh unit with data;

said external memory units forming a frame buffer, said frame buffer storing at least one of an RGBα, a depth and a stencil value for performing at least one of volume rendering and geometry rendering, said frame buffer being interleaved across said plurality of memory units, in such a way that ordered access by the Cube-5 volume rendering pipelines allows full usage of the bandwidth of the external memory units; and

instructing said geometry pipeline of said buffer interleaving structure such that said geometry pipeline correctly accesses said frame buffer.

55.   A method for connecting a Cube-5 volume rendering pipeline to a geometry pipeline, said Cube-5 rendering pipeline including a frame buffer, said method comprising the steps of:

transmitting geometry data from said geometry pipeline to said Cube-5 pipeline, said geometry data including images of rasterized geometries;

combining said geometry data in said Cube-5 pipeline;

storing said combined geometric data in said frame buffer.

56.   The method of Claim 55, further comprising the step of:

encoding said geometry data in run length encoded (RLE) format.

57.   The method of Claim 56, further comprising the steps of:

providing a run length encoded rasterization engine;

connecting said rasterization engine to said frame buffer.

FIG-1 PRIOR ART

........> VOLUME DATA FLOW
& PROCESSING

------> IMAGE FLOW
& PROCESSING

MONITOR

GRAPHICS CARD
(BUFFER)

CPU

MEMORY

DISK

BUS

FIG-2

FIG-3

GEOMETRY ENGINE

30

10

24

MEMORY

24

MEMORY

26

FEEDBACK

CUBE-5 CHIP

28

CUBE-5 CHIP

28

TO FRAME BUFFER

HOST

32

34

FIG-4

FIG-5

TEXTURE COORDINATES
GEOMETRY SLICES

52

ADDRESS
GENERATION
AND CONTROL
62

GEOMETRY
ENGINE
76

24

3D
MEMORY

MEMORY
CONTROL
78

DOUBLE SLICE
VOXEL FIFO
64

54

TriLin

GRADIENT
56

70

TEXTURE
MAP
BYPASS

58

SHADER

72

60

COMPOS

BiLin

MULTIPLE
VOLUMES
FEEDBACK

66

COMPOSITING
FEEDBACK

TEXELS

74

40

2D MEMORY

COMPOS
BUFFER
RGBAZ

68

RGBAZ

IMAGE
BASED
RENDERING
FEEDBACK

44

WARP

LOD SCALING
FEEDBACK

TO FRAME BUFFER

FIG-6A

2^3 VOXEL
MINIBLOCK
(8 16 - bit)

2^2 TEXEL
TILE
(4 32 - bit)

RG BA

FIG-6B

TEXEL		VOXEL	
0	RG	0	
	BA	4	
1	RG	1	
	BA	5	
2	RG	2	
	BA	6	
3	RG	3	
	BA	7	

FIG-7

SOURCE IMAGE

TARGET IMAGE

FORWARD MAPPING

SPECIAL SCANLINE

SAMPLE POINTS

FIG-8

FIG-9

FIG-10

FIG-11

FIG-12

MAPPED
PIXEL GRID

116

114

SOURCE IMAGE (conic)

118

r_1

r_2

BACKWARD MAPPING

PIXEL
GRID

x

y

TARGET IMAGE (circle)

FIG-13

FIG-14

FIG-15

17/46

FIG-16
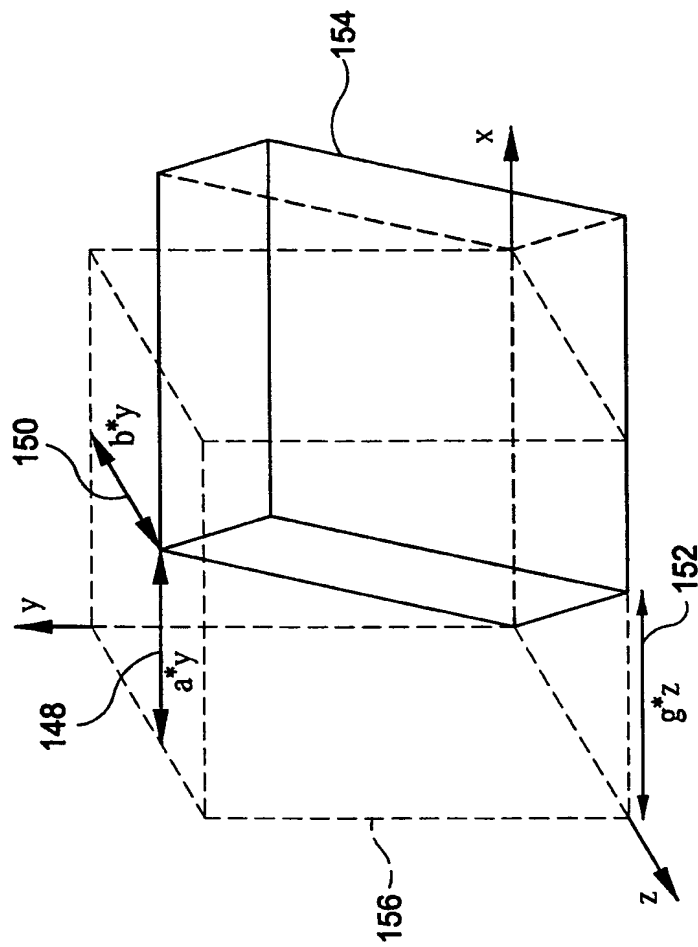
FIG-17

Y - SLICE

X BEAM

158

158'

x

y

z

Z - SLICE

FIG-18A PRIOR ART

7 RAYS

160
160
160
160
160
160
166
164
162

VOXELS NOT CONTRIBUTING

FIG-18B PRIOR ART

FIG-19

FIG-20A

BASEPLANE

REGION
BOUNDARIES

BASEPLANE PIXELS

FIG-20B

FIG-21

FIG-22

BASEPLANE

eye$_z$ = 3 ─ 196

REGION BOUNDARIES

VOLUME

• VOXEL COORDINATES

▲ RESAMPLING FILTER WEIGHTS

X RAY SAMPLE LOCATIONS
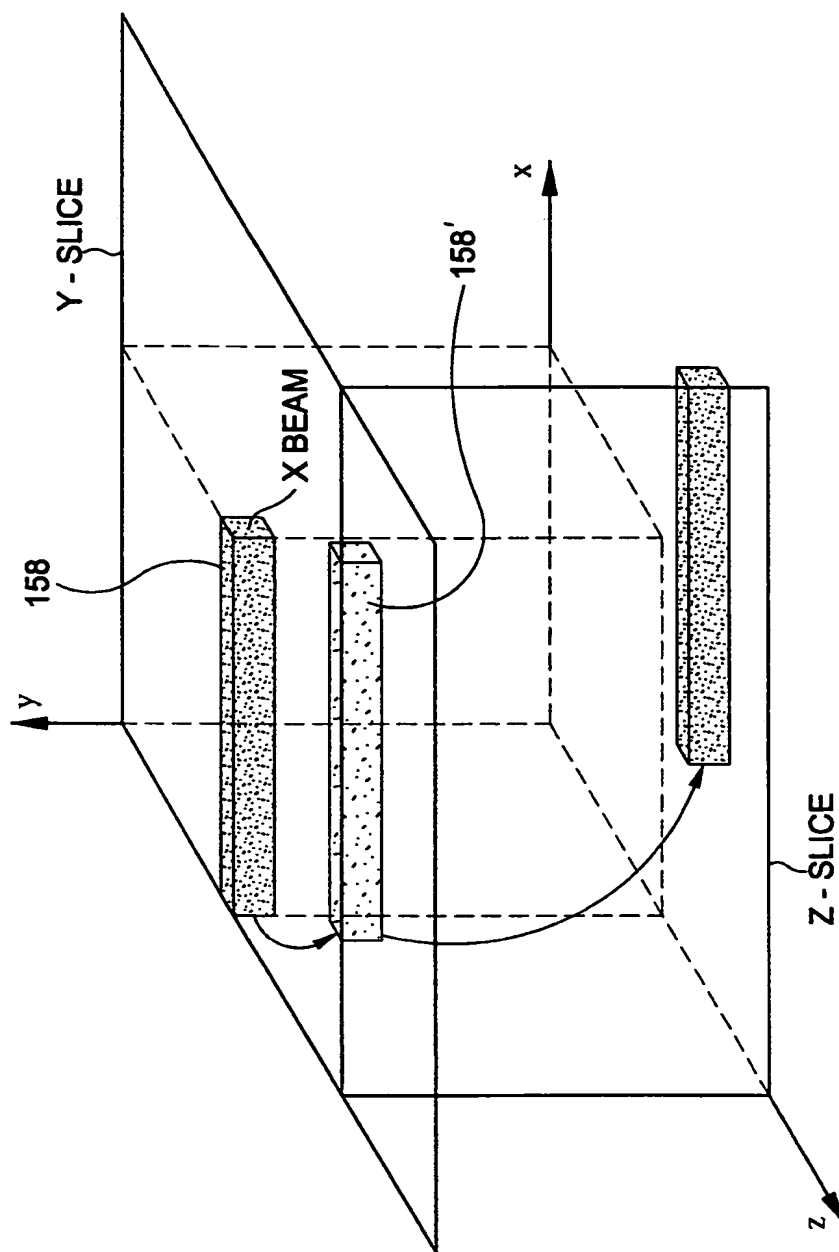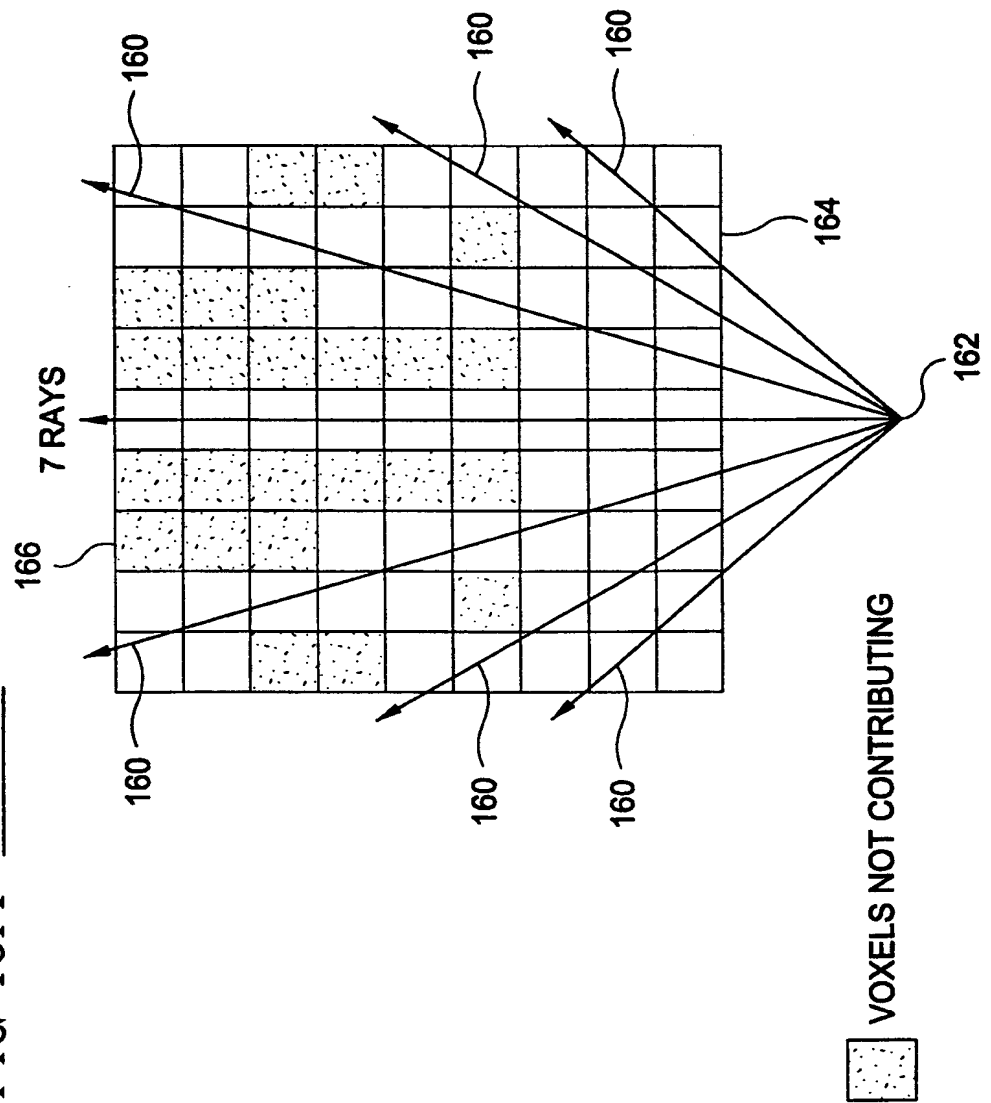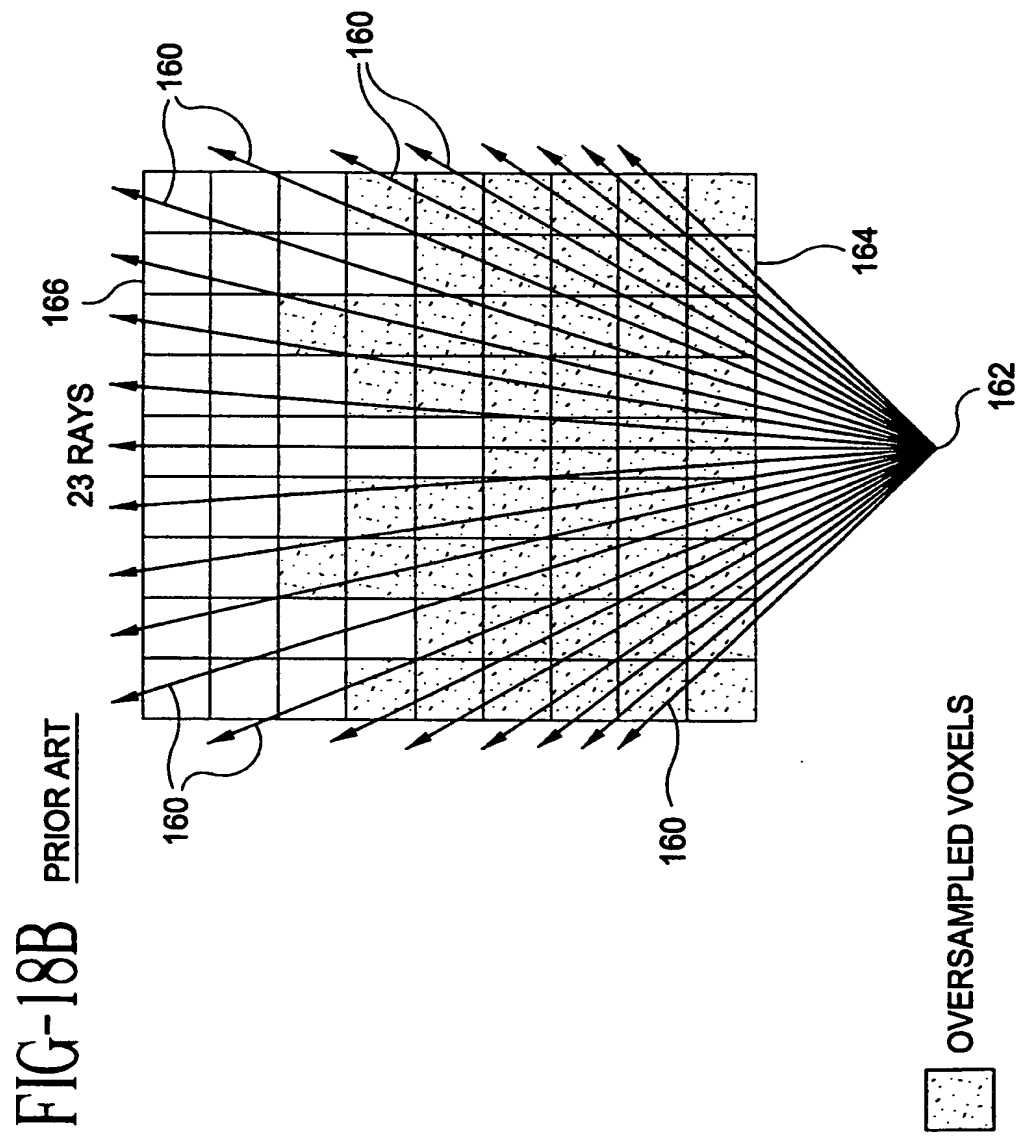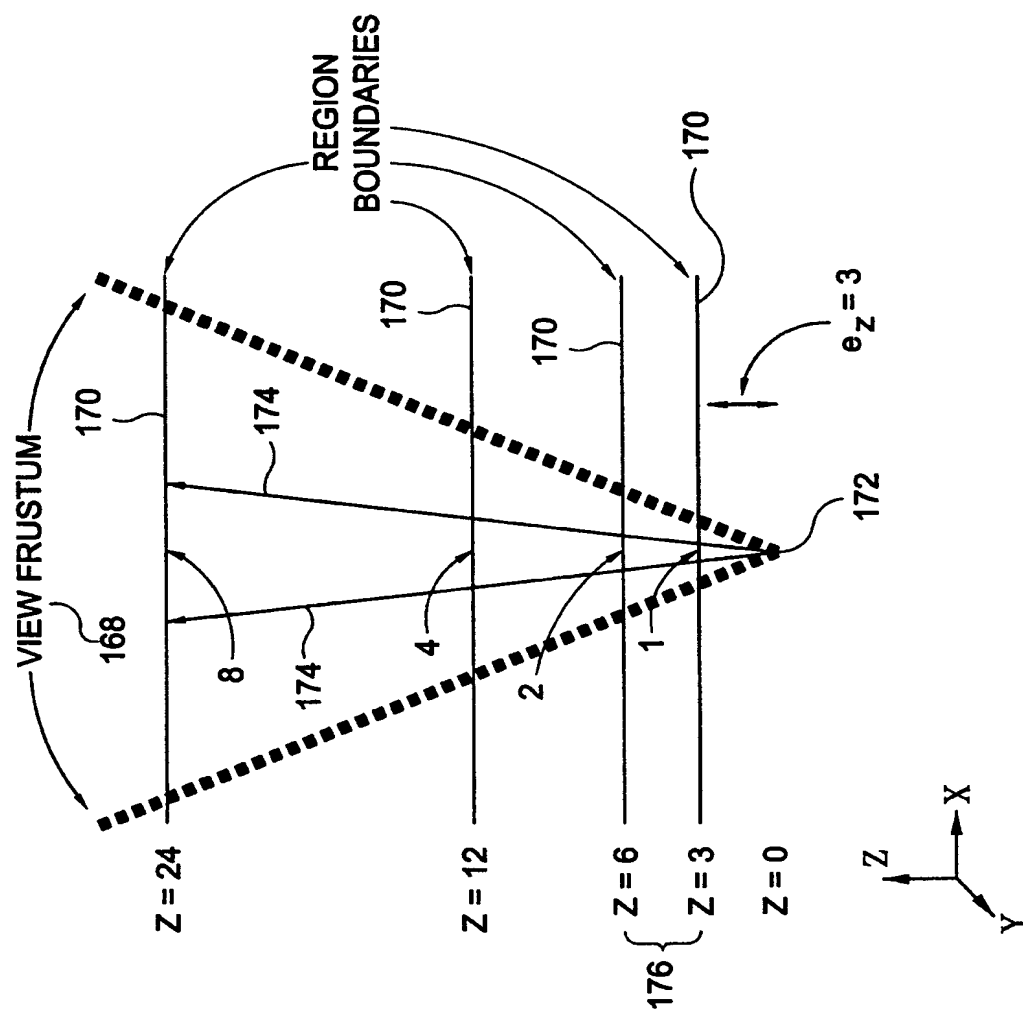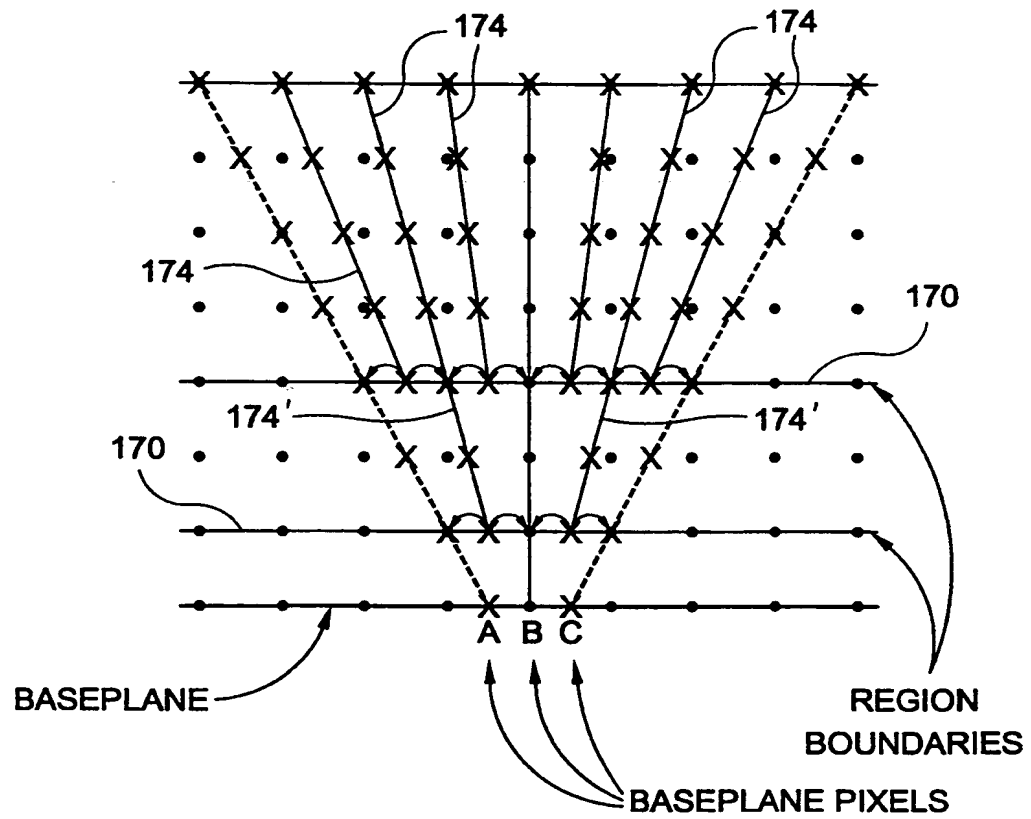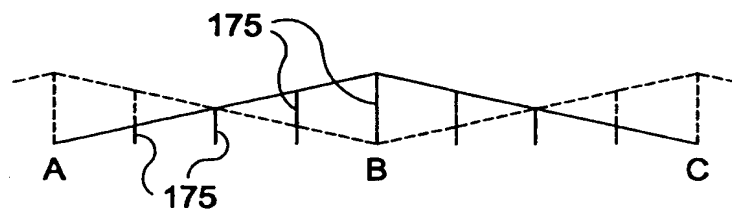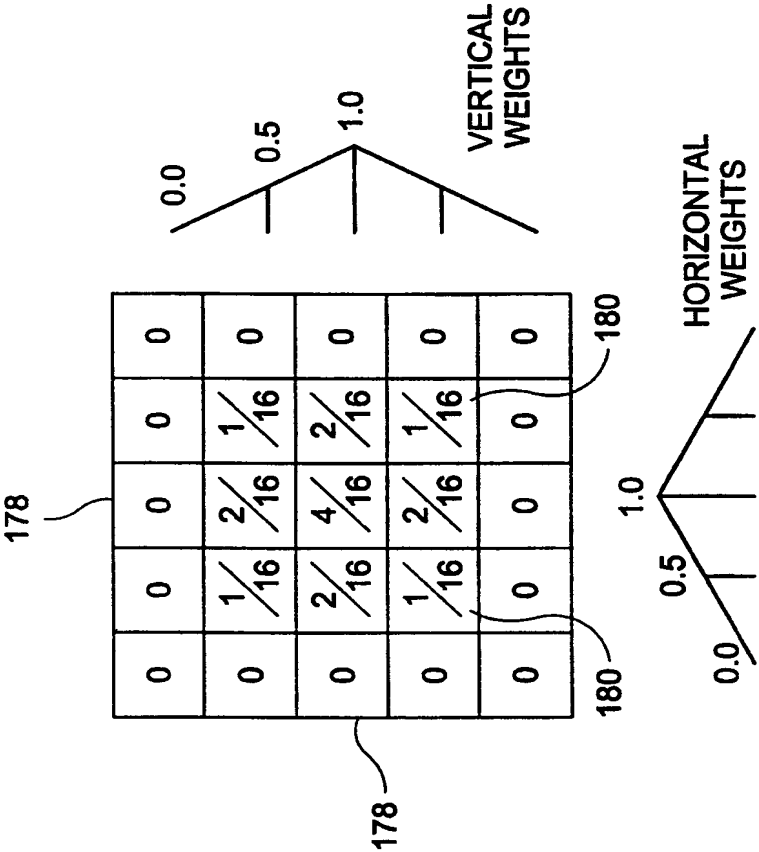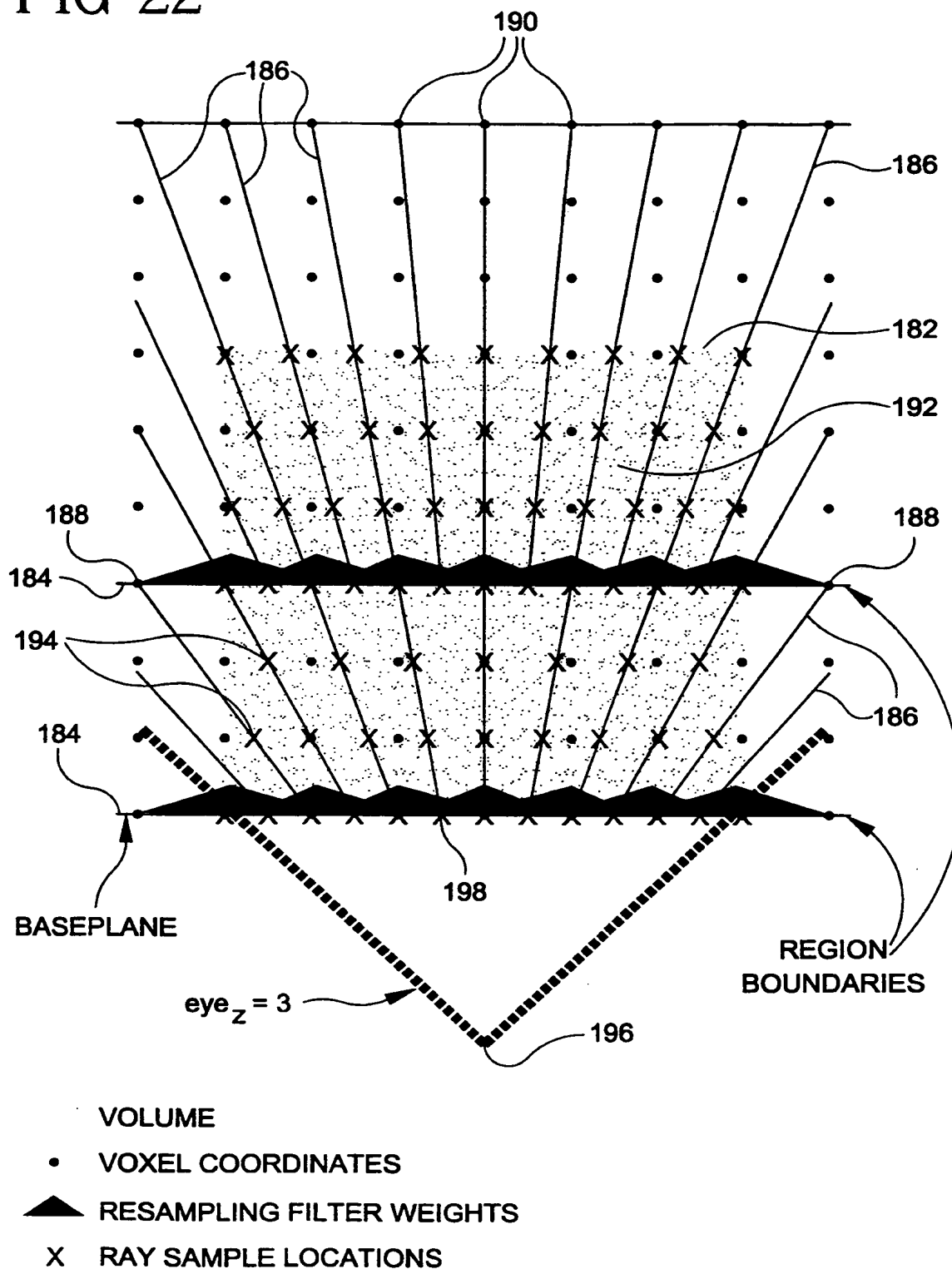
FIG-23

Compute Z-position of Eye in Voxel Units
Compute Exponential Region Boundaries
for REGION = MAXREGION to 0
  for SLICE = MAXSLICE[REGION] to MINSLICE[REGION]
    Interpolate Samples for this slice (Bilin)
    Shade and Classify Samples
    Composite Samples onto Rays in Buffer
  end for
  if not frontmost REGION
    Downsample Rays in Compositing Buffer with
       Bartlett Filter
    end if
end for
Warp Baseplane to Final Image plane

FIG-24

REGION 2

REGION 1

BARTLETT FILTER WEIGHTS
FOR RAY RESAMPLING

FIG-25

x - WEIGHTS : -1 0 1
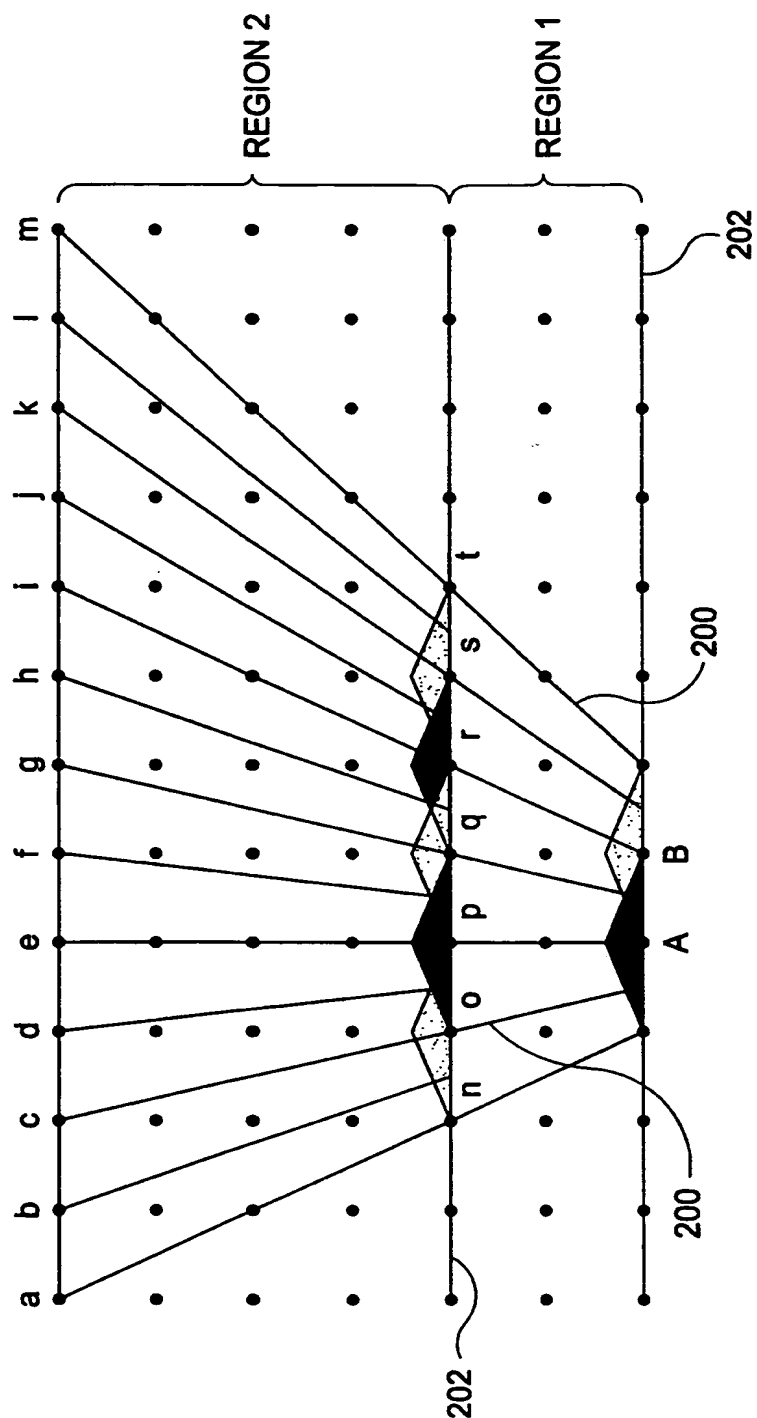
y - WEIGHTS : 1 w 1

z - WEIGHTS : 1 w 1

RESULTING EFFECTIVE 3X3X3 FILTER :

$$
\begin{array}{ccc}
-1 & 0 & 1 \\
-w & 0 & w \\
-1 & 0 & 1
\end{array}
\qquad
\begin{array}{ccc}
-w & 0 & w \\
-w^2 & 0 & w^2 \\
-w & 0 & w
\end{array}
\qquad
\begin{array}{ccc}
-1 & 0 & 1 \\
-w & 0 & w \\
-1 & 0 & 1
\end{array}
$$

FIG-26

FIG-27

NUMBER OF VERTICES WITHIN SLICE	RESULTING NUMBER OF TRIANGLES
0	2
1	1
1	3
2	2

FIG-28

BUCKETS	MEMBER TRIANGLES
S1	T1
S2	T1,T2
S3	T1,T2
S4	T1,T3,T4
S5	T1,T4
S6	T1,T4

FIG-29

SHEARED (PRE-WARPED) GEOMETRY

ACTUAL GEOMETRY

GEOMETRY VIEW VOLUME

α

α

228

230

232

234

234
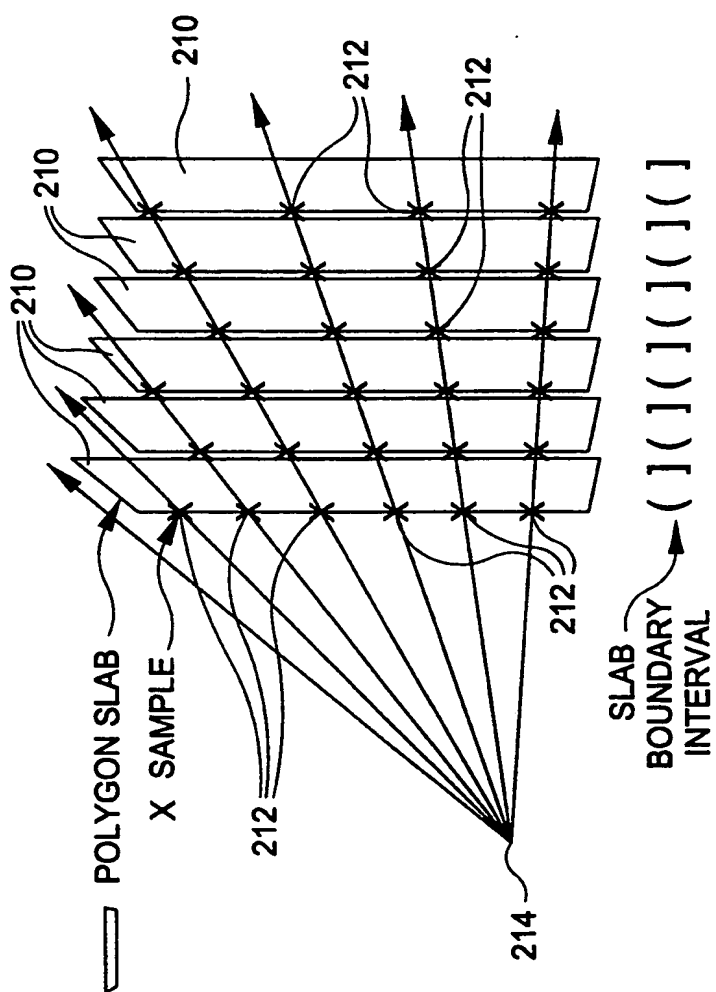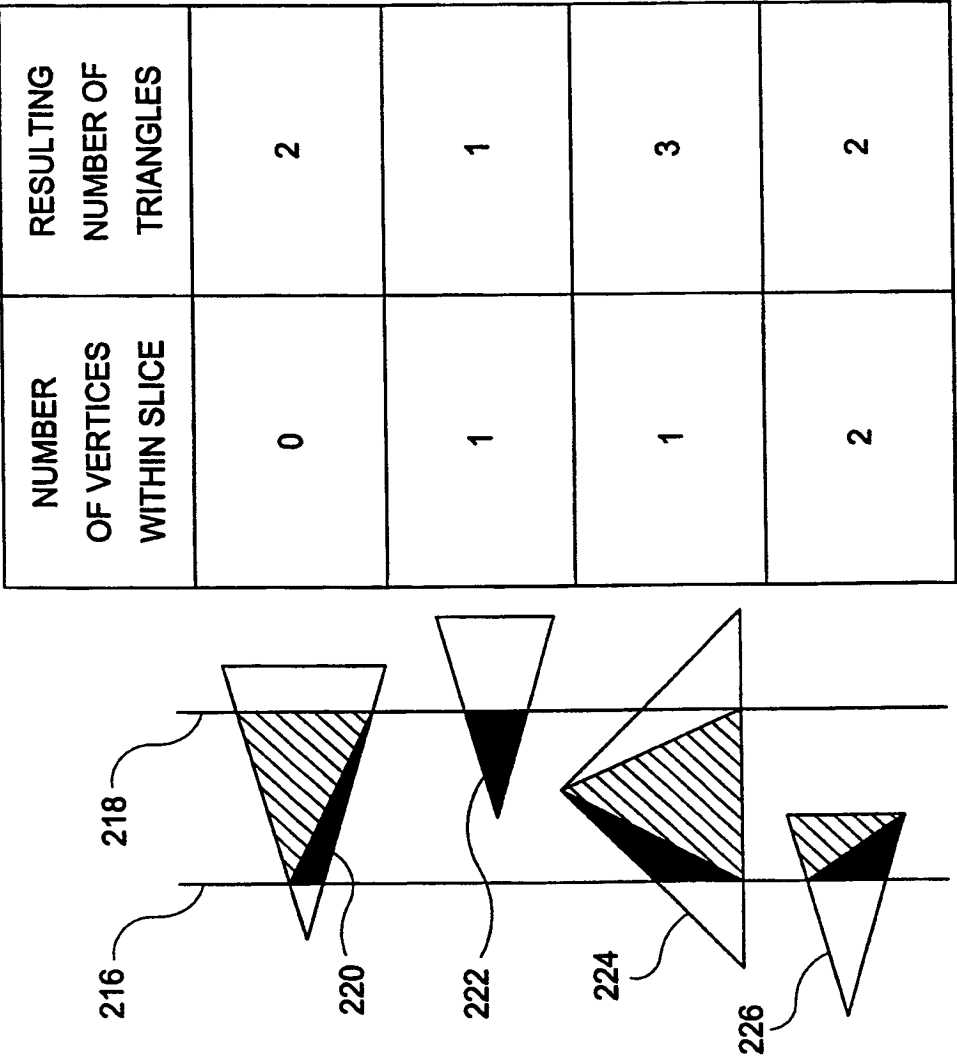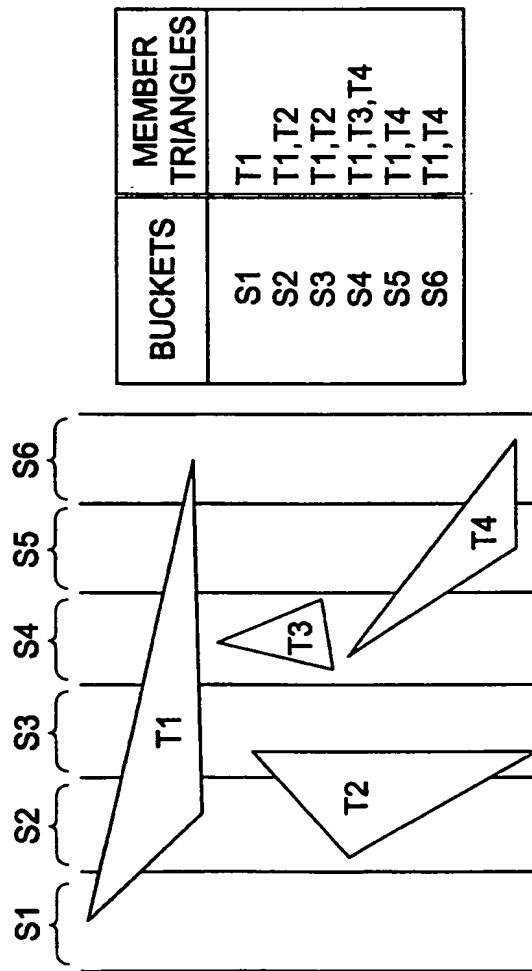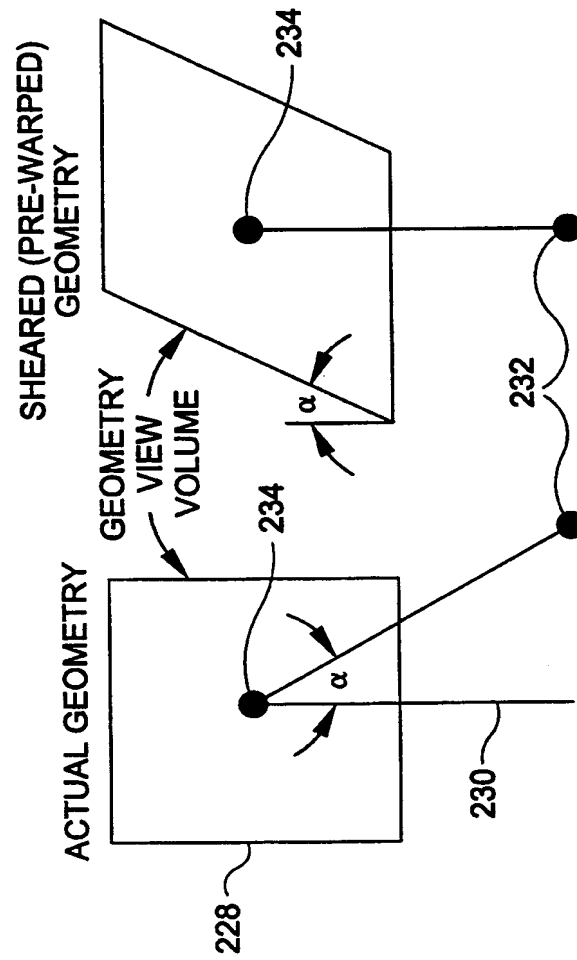
FIG-30
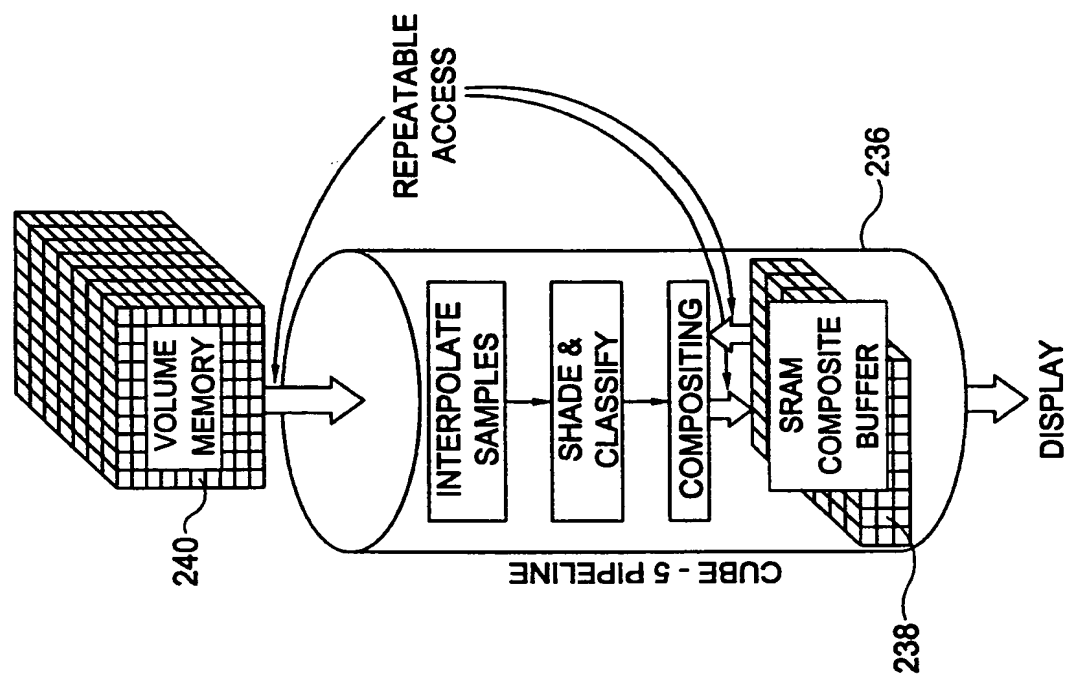
FIG-31 PRIOR ART
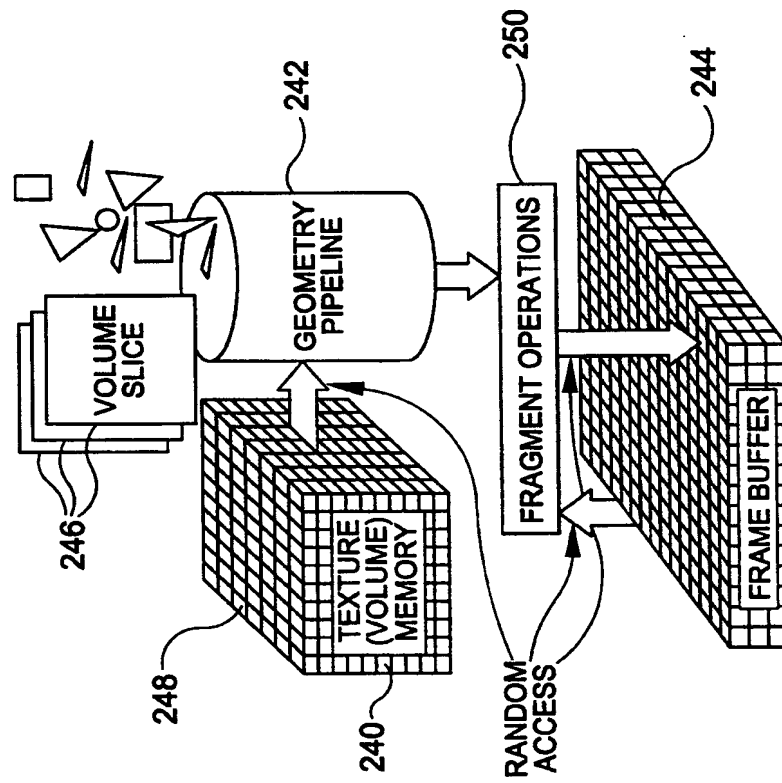
FIG-32
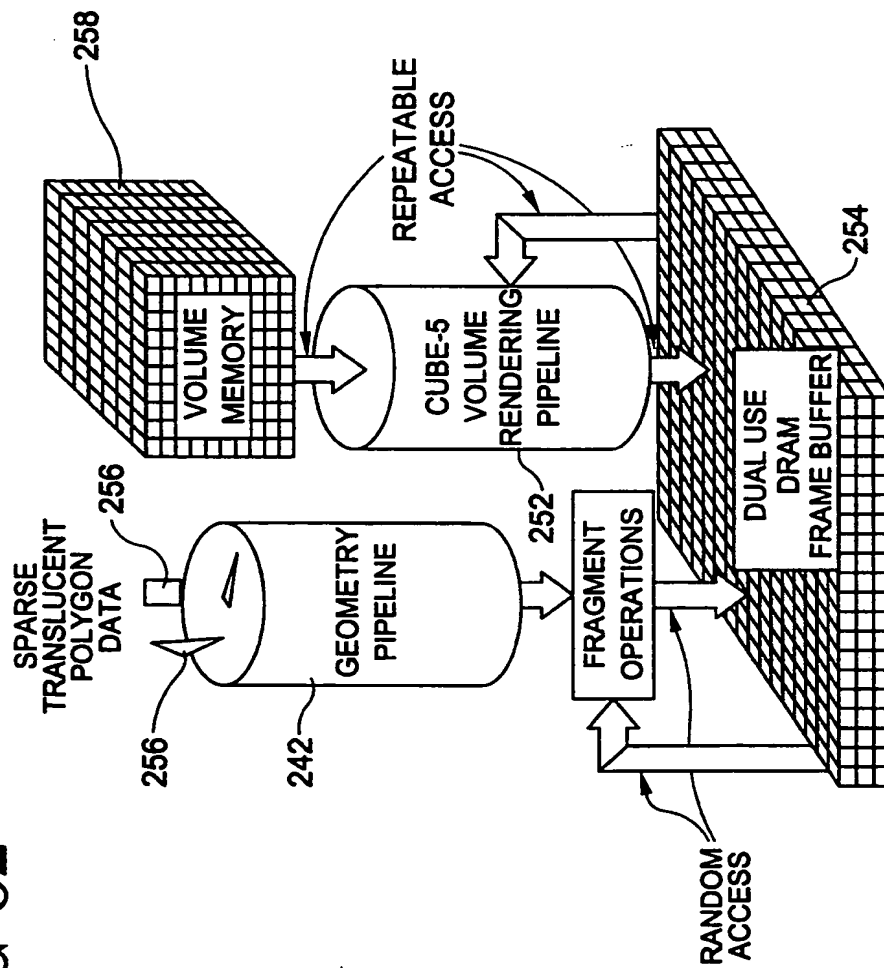
FIG-33
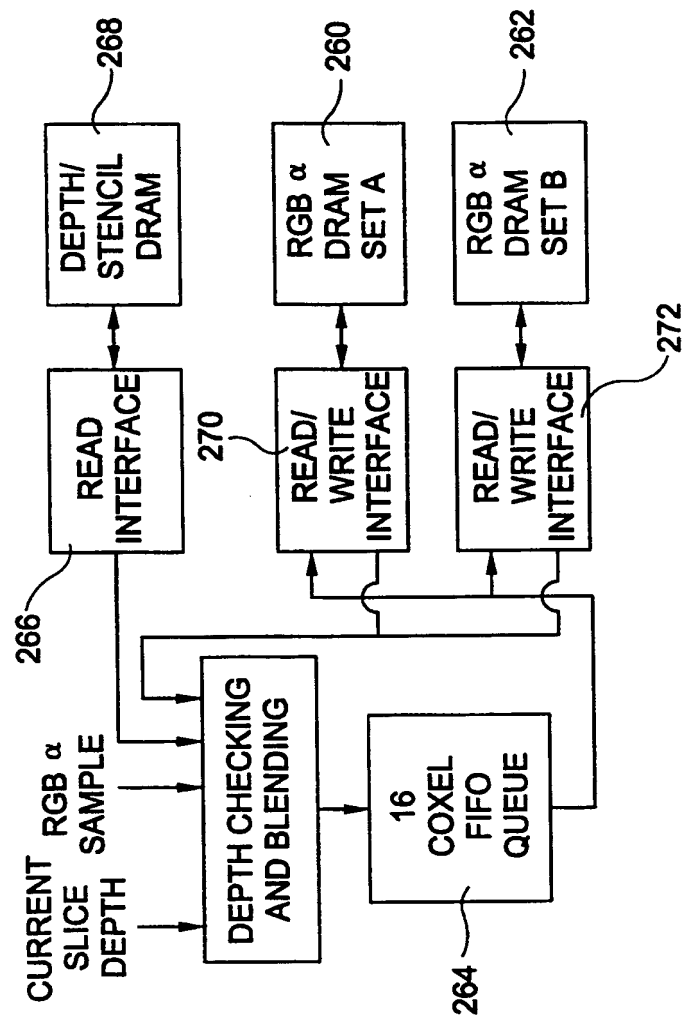
FIG-34
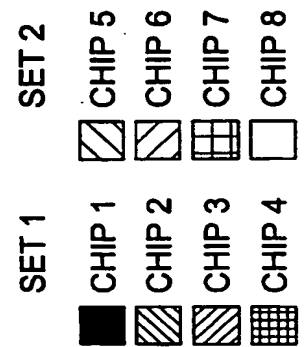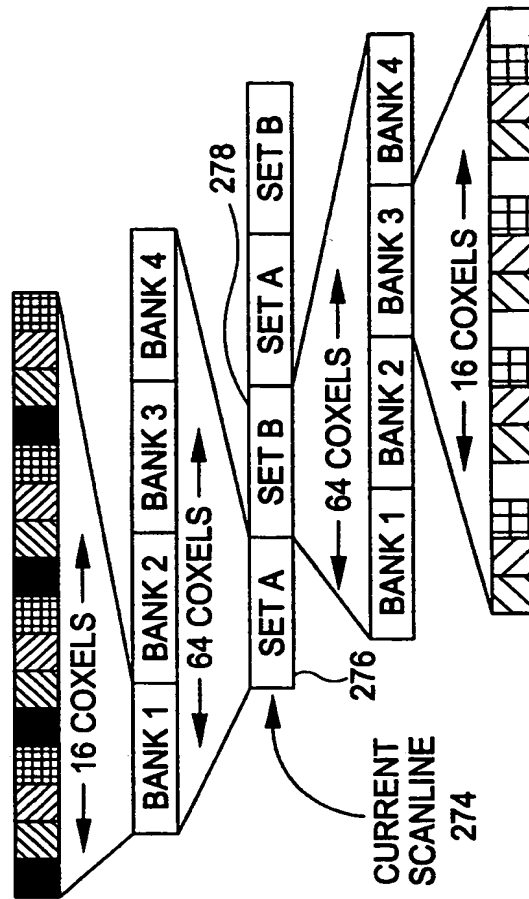
FIG-35
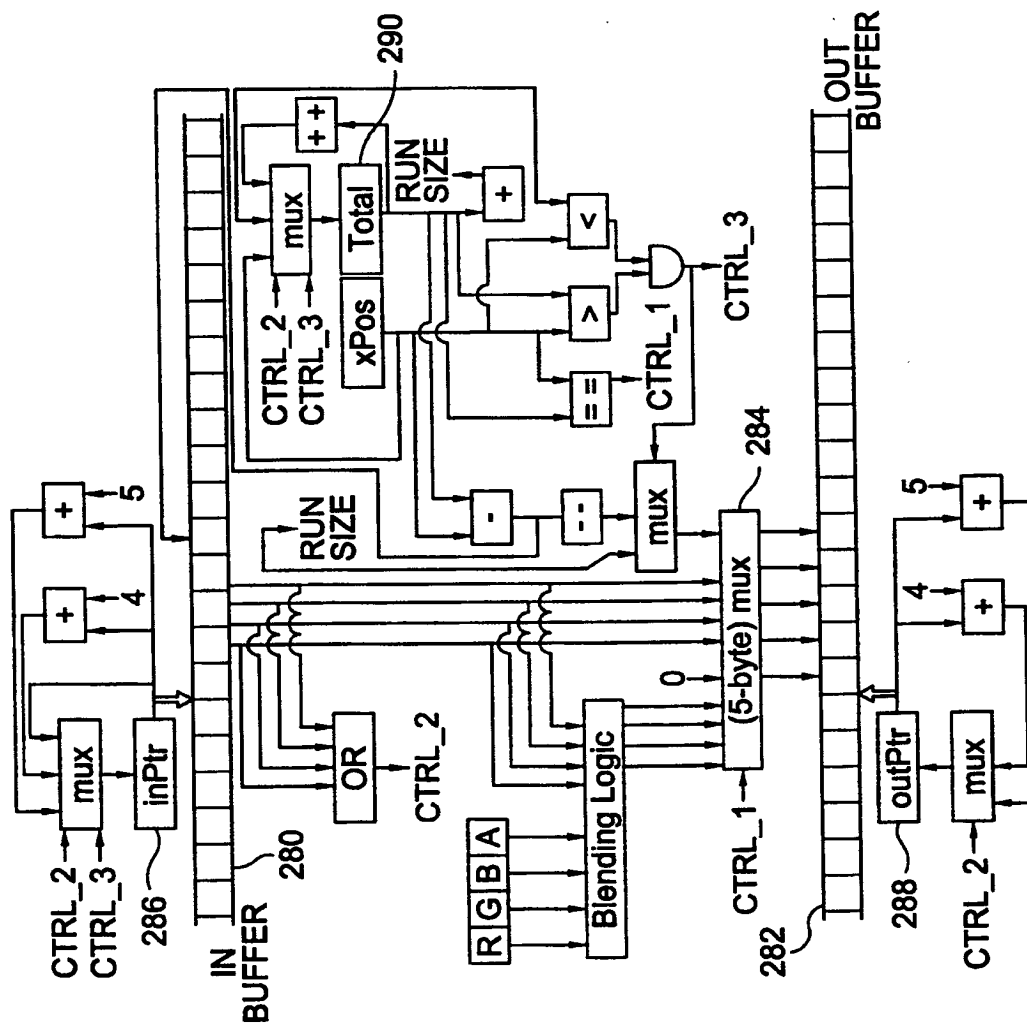
FIG-36

```
RLE_AddFragment (xPos, yPos, RGBA) {
    tmp = nextFreeScanline() ;
    RLE_AddPixelToScanline(data[yPos], xPos, RGBA, tmp) ;
    freeScanLine (data [yPos] ) ;
    data [yPos] = tmp ;
}

RLE_AddPixelTo Scanline(in, xPos, RGBA, out) {
    total = 0 ;
    inPtr = 0 ;
    outPtr = 0 ;
    while(total < lineWidth) {
        if (total == xPos) {
            out [outPtr: :outPtr+3] = BLEND (RGBA, in [inPtr: :inPtr+3]);
            outPtr +=4;
            total++;
            if(in[inPtr: :inPtr+3] == 0)
                in [inPtr+4] - - ;
            else
                inPtr +=4;
        }

        out [outPtr: :outPtr+3] = in [inPtr: :inPtr+3] ;
        if (in [inPtr: :inPtr+3] == 0) {
            if (total < xPos && total+in[inPtr+4] > xPos) {
                out [outPtr+4] = xPos - total -1 ;
                outPtr +=5 ;
                in [inPtr+4]  -= xPos-total ;
                total = xPos ;
            } else {
                out [outPtr+4] = in[inPtr+4] ;
                total += in [inPtr+4] ;
                outPtr +=5 ;
                inPtr  +=5 ;
            }
        } else {
            total ++ ;
            outPtr +=4 ;
            inPtr  +=4 ;
        } // endif run-of-zeroes
    } // endwhile still within scanline
}
```
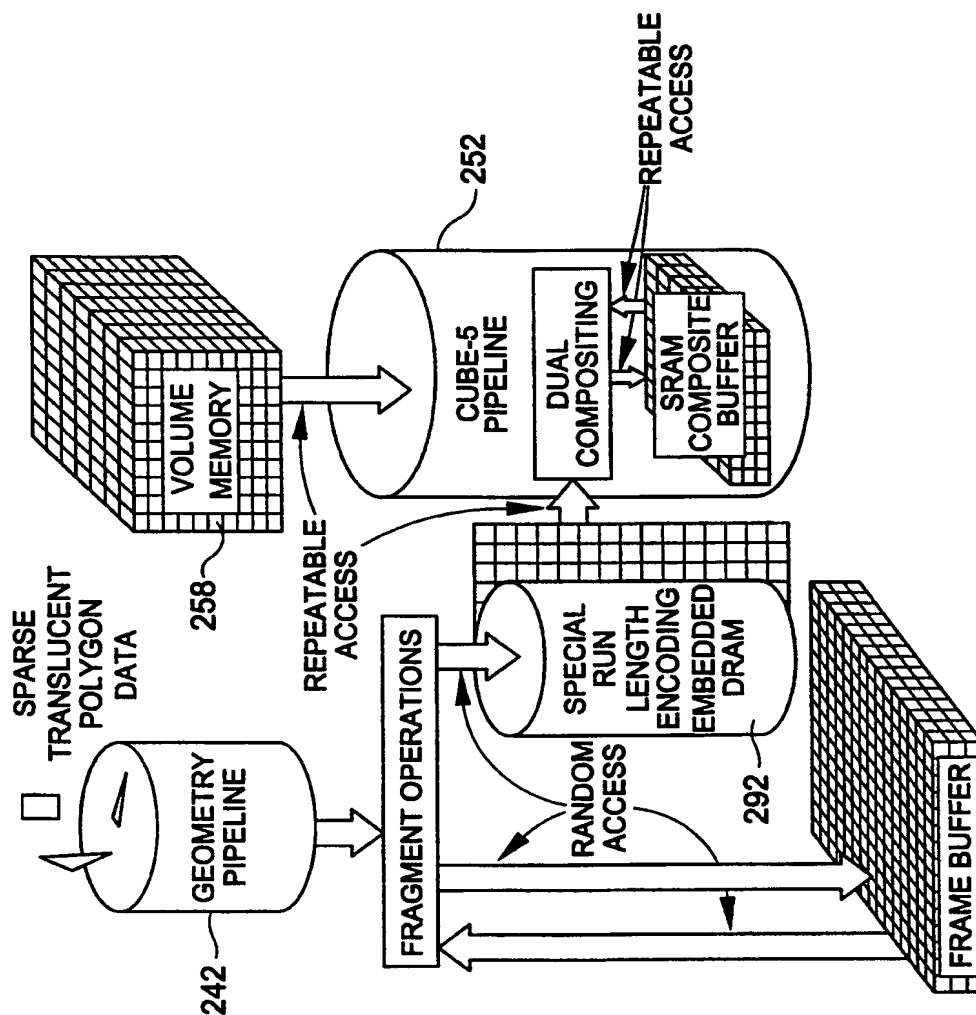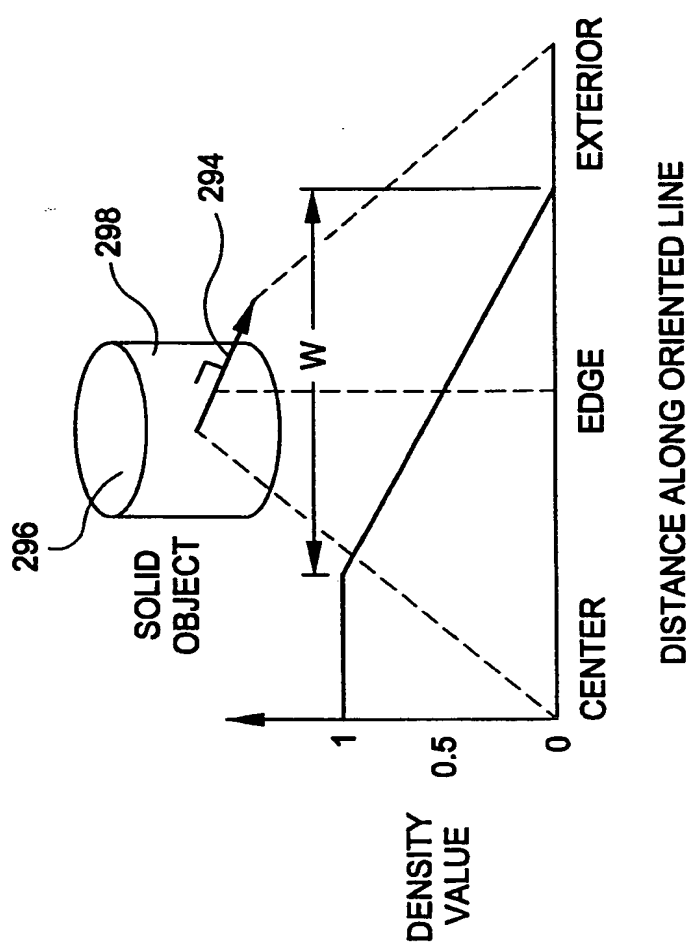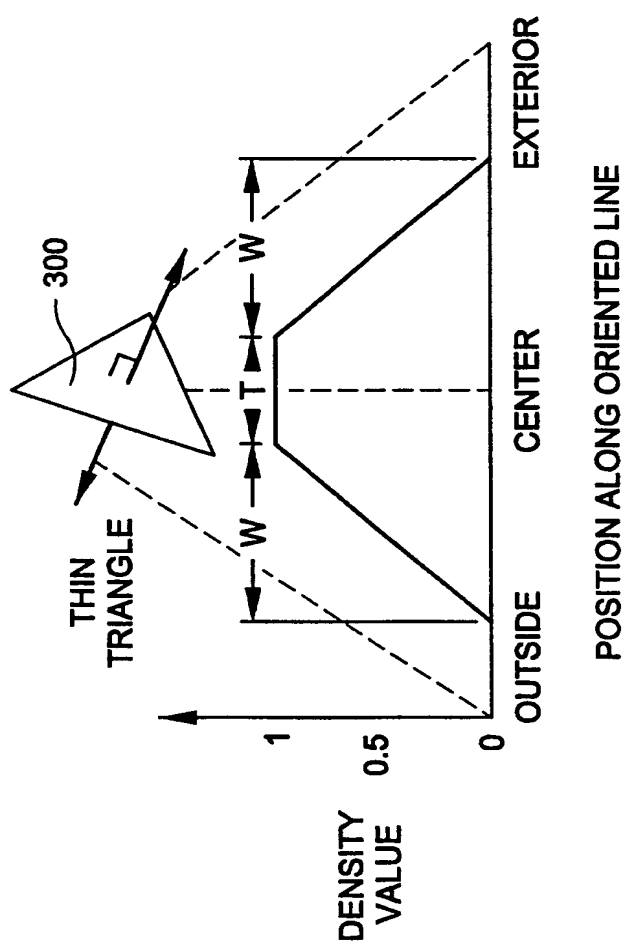
FIG-37

FIG-38

SOLID
OBJECT

298

296

294

W

DENSITY
VALUE

1

0.5

0

CENTER   EDGE   EXTERIOR

DISTANCE ALONG ORIENTED LINE

FIG-39

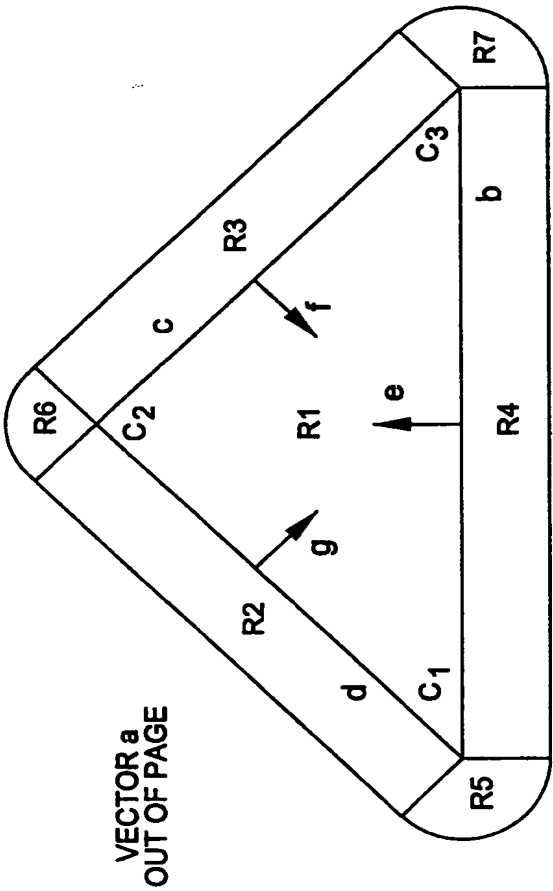FIG-40

FIG-41

```
Define Plane(A, B, C, D);
Find triangle bounding box(bb);
Dist = A x bb.min.x + B x bb.min.y + C x bb.min. z +
D;
xStep = A;
yStep = B - A x bb.width;
zStep = C - B x bb.height - A x bb.width;
For z = bb.min.z to bb.max.z with unit steps
    For y = bb.min.y to bb.max.y
        For x = bb.min.x to bb.max.x
            store f(Dist) in [x,y,z] (voxelize)
            Dist = Dist + xStep;
        end For
        Dist = Dist + yStep;
    end For
    Dist = Dist + zStep;
end For
```
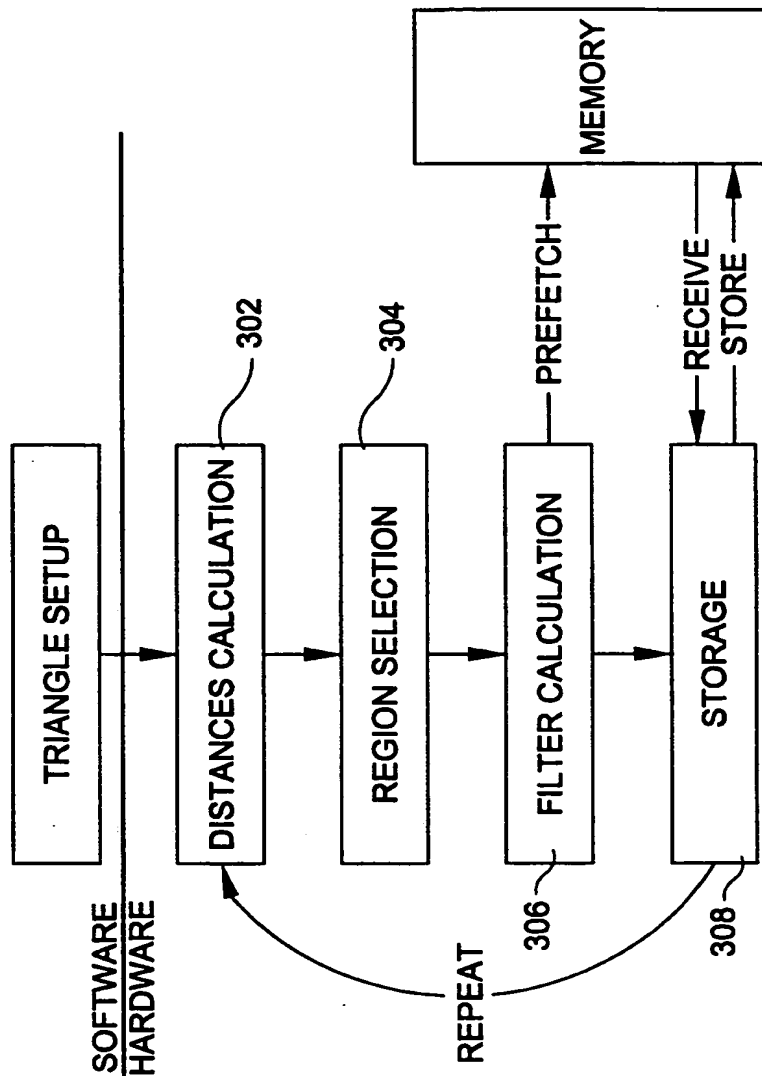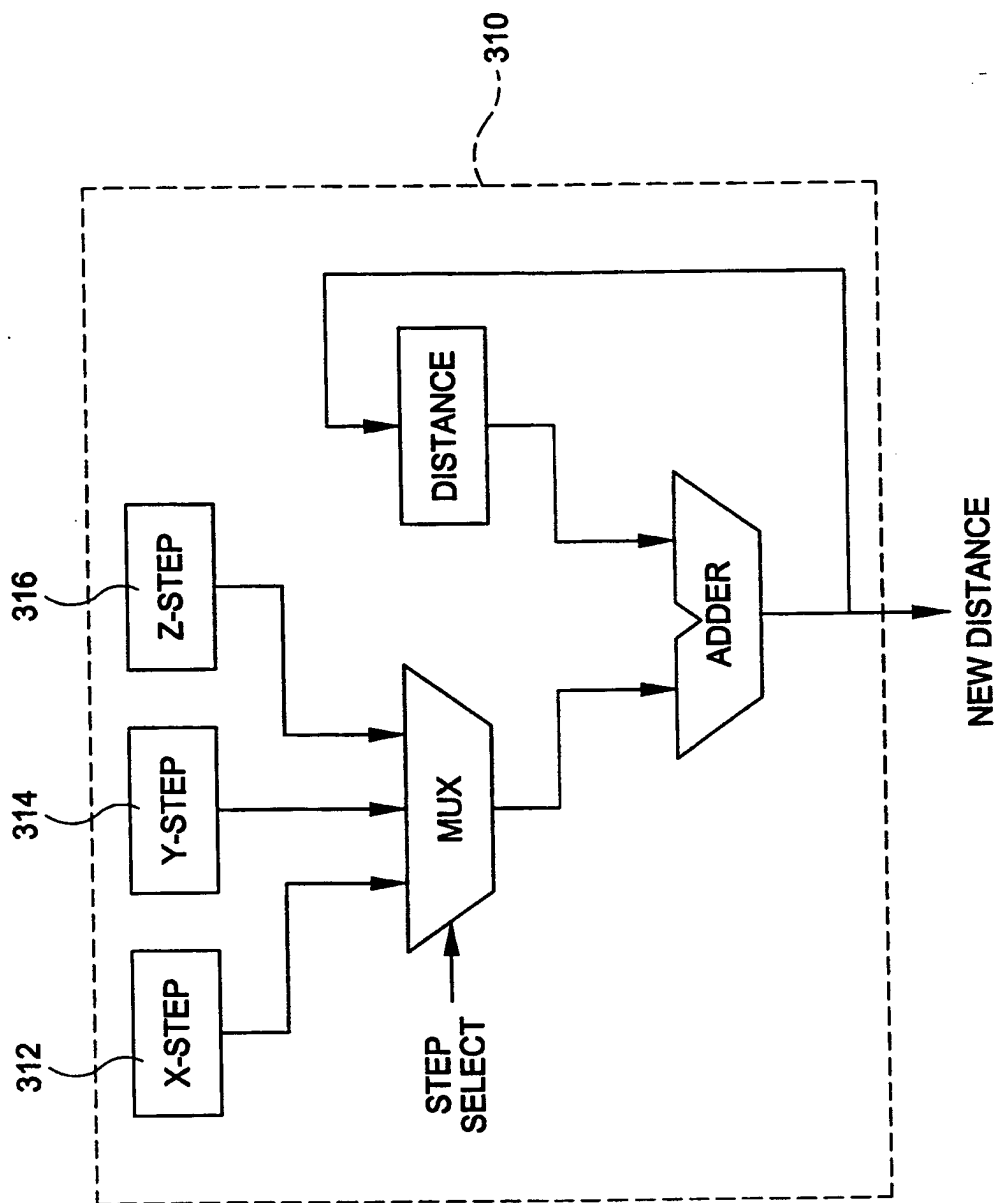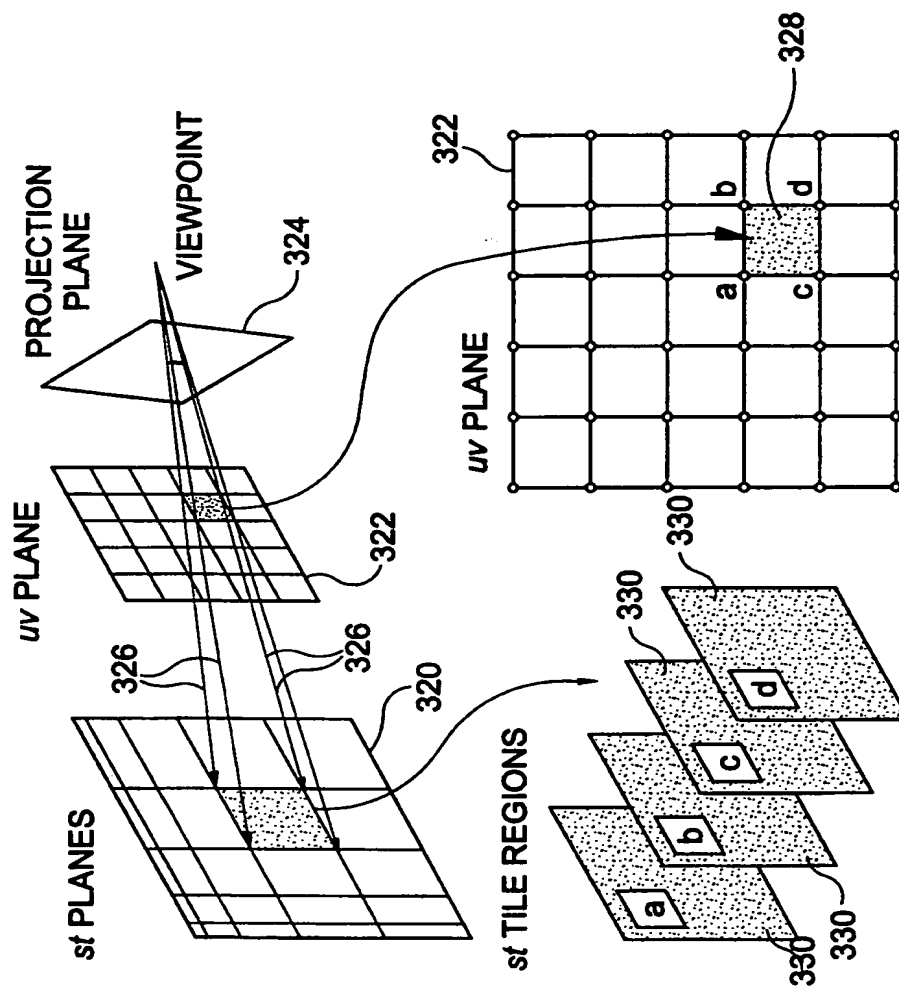
FIG-42

TRIANGLE SETUP

SOFTWARE
HARDWARE

DISTANCES CALCULATION   302

REGION SELECTION   304

FILTER CALCULATION   306

PREFETCH

STORAGE   308

RECEIVE

STORE

MEMORY

REPEAT

FIG-43

FIG-44

INTERNATIONAL SEARCH REPORT

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06T 17/00
US CL :345/419, 424

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 345/418, 419, 424, 420, 425

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,724,493 A (HOSOYA et al) 03 March 1998.	1-57
A	US 5,760,781 A (KAUFMAN et al) 02 June 1998.	1-57
A,P	US 5,847,711 A (KAUFMAN et al) 08 December 1998	1-57
A,P	US 5,850,226 A (NAGASAWA et al) 15 December 1998	1-57
A	US 5,877,779 A (GOLDGERG et al) 02 March 1999	1-57

☐ Further documents are listed in the continuation of Box C.   ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance		
"E"	earlier document published on or after the international filing date	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed	"&"	document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
12 SEPTEMBER 1999	15 october 1999

| Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No.   (703) 305-3230 | Authorized officer
CLIFF N~~~~~O R. M~~~~~
Telephone No.   (703) 305-9594 |

Form PCT/ISA/210 (second sheet)(July 1992)*